Pushing the Boundaries of Deep Parsing

A thesis presented
by

Andrew MacKinlay

to
The Department of Computing and Information Systems
in total fulfillment of the requirements
for the degree of
Doctor of Philosophy

The University of Melbourne
Melbourne, Australia
September 2012

Declaration

This is to certify that:

(i) the thesis comprises only my original work towards the PhD except where indicated in the Preface;

(ii) due acknowledgement has been made in the text to all other material used;

(iii) the thesis is fewer than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Signed: _____ Date: _____

Thesis advisor(s)
Timothy Baldwin
Rebecca Dridan

Author
Andrew MacKinlay

Pushing the Boundaries of Deep Parsing

Abstract

I examine the application of deep parsing techniques to a range of Natural Language Processing tasks as well as methods to improve their performance. Focussing specifically on the English Resource Grammar, a hand-crafted grammar of English based on the Head-Driven Phrase Structure Grammar formalism, I examine some techniques for improving parsing accuracy in diverse domains and methods for evaluating these improvements. I also evaluate the utility of the in-depth linguistic analyses available from this grammar for some specific NLP applications such as biomedical information extraction, as well as investigating other applications of the semantic output available from this grammar.

Contents

Citations to Previously Published Work

Large portions of Chapter 4 have appeared in the following paper:

> Andrew MacKinlay, Rebecca Dridan, Dan Flickinger and Timothy Baldwin. 2011. Cross-Domain Effects on Parse Selection for Precision Grammars. In *Research on Language and Computation*, Vol 8(4) pp 299–340.

Large portions of Chapter 5 have appeared in the following paper:

> Andrew MacKinlay, Rebecca Dridan, Dan Flickinger, Stephan Oepen and Timothy Baldwin. 2011. Treeblazing: Using External Treebanks to Filter Parse Forests for Parse Selection and Treebanking. In *Proceedings of the 5th International Joint Conference on Natural Language Processing (IJCNLP '11)*, pp 246–254

Large portions of Chapter 6 have appeared in the following papers:

> Andrew MacKinlay, David Martinez and Timothy Baldwin. 2009. Biomedical Event Annotation with CRFs and Precision Grammars. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, pp 77–85.

> Andrew MacKinlay, David Martinez and Timothy Baldwin. 2011. A Parser-based Approach to Detecting Modification of Biomedical Events. In *Proceedings of the ACM Fifth International Workshop on Data and Text Mining in Biomedical Informatics (DTMBIO '11)*, pp 51–58

> Andrew MacKinlay, David Martinez and Timothy Baldwin. 2012. Detecting Modification of Biomedical Events Using a Deep Parsing Approach. In *BMC Medical Informatics and Decision Making*, Vol 12(Suppl 1), pp S4

In addition, the work in Chapter 7 is based on a unpublished draft co-authored with Ann Copestake and Dan Flickinger. In almost all cases I was the primary author and the core implementation was substantially my own — the one exception being MacKinlay *et al.* (2009), where one subsystem was primarily the work of David Martinez, but this portion of the paper has not been included as an original contribution of this thesis. In each case, I have only included and adapted prose from these publications which was primarily written by me.

Acknowledgments

There are many people who have contributed in small and large ways to this thesis, both directly and indirectly. In the indirect sense, there are of course all the people around me who have been supportive throughout my study. My parents Alistair and Ros laid the groundwork in bringing me up in a positive environment and setting me on the educational path which led me to the University of Melbourne for my undergraduate degrees and then my postgraduate study. My partner Kate has also been amazingly supportive and tolerant during my time studying.

More directly, I am grateful to the funding bodies which contributed to my scholarships which have supported me over these four years — particularly National ICT Australia. Funding for various interstate and international trips also came from various sources — in particular, NICTA (especially the BioTALA group), the University of Melbourne, The University of Oslo, and Google — and these trips have also been extremely valuable for my education.

During my PhD, I was very fortunate to be able to make extended visits to several overseas institutions to work with some extremely talented and helpful researchers. These visits were incredibly useful and productive, both in terms of life experience, and extending and broadening the scope of my research, and I hope that my visits made some contribution in return. The thesis would have turned out very differently if I had not had these opportunities — indeed, each of these research visits formed the basis of a chapter in this thesis. Specifically, I would like to thank my primary hosts at each of the institutions I visited, who generously gave up their time to make the visits possible. At Cambridge, I was hosted by Ann Copestake from the Natural Language and Information Processing (NLIP) group in the Computer Laboratory. At Stanford, I was hosted by Dan Flickinger, from the Center for Study of Language and Information (CSLI). At the University of Oslo, I was hosted for a regrettably brief visit by Stephan Oepen from the Department of Informatics (IFI). All of these researchers have continued to provide the same high quality of academic and technical advice after the end of the respective visits. Dan deserves an extra mention since he generously agreed to add to his already formidable workload in the thankless and time-consuming task of treebanking for the research described in Chapter 5. There was another more global community of people who I was also lucky to interact with. The group of researchers involved with the DELPH-IN consortium (including all of my aforementioned hosts as well as many others) has always been helpful and open. It was also a highly valuable experience to meet many of these researchers in person in Japan during the earlier stages of my thesis.

This thesis was shaped very positively by a great many interactions with other researchers. Whether these interactions took the form of collaboration, technical advice, someone generously taking time out of their busy schedule to have a conversation about my work and offer helpful suggestions, or simply having a conversation which expanded my knowledge of the research field, they all helped in the creation of the thesis. It is probably already clear that my hosts provided much assistance along these lines, but many other people made smaller but still important contributions. At each institution I have visited, I have found communities of talented academics,

postdocs and postgraduate students, and engaging with them was highly rewarding both academically and socially. I would like to thank all of the NLP and Computational Linguistics students and researchers from the NLIP group in Cambridge, CSLI at Stanford, IFI at the University of Oslo, and of course the Language Technology group from the Department of Computer Science and Software Engineering at The University of Melbourne, where I spent the majority of my time during the PhD (despite the seemingly full travel schedule). In addition I would like to thank visitors to these respective locations who met with me, those who I interacted with productively during conferences, and those who were kind enough to provide me with useful data. Additionally many of the reviewers for the various publications which were adapted into this thesis provided insightful comments which in turn improved the thesis itself.

Particular people from the above categories deserve to be named individually. This non-exhaustive list (beyond those I have already thanked) of people who were particularly forthcoming with advice or data includes Peter Adolphs, Francis Bond, Jari Björne, Phil Blunsom, Ekaterina Buyko, Lawrence Cavedon, Trevor Cohn, Udo Hahn, Mark Johnson, David Martinez, Diarmuid Ó Seaghdha, Jonathon Read, Ulrich Schäfer, Erik Velldal, Karin Verspoor, Andreas Vlachos, Willy and Dominic Widdows. This list will inevitably have missed someone who deserves to be included — to any researchers who I have not specifically mentioned, in the unlikely event you see this thesis, I hope you feel included by my more general acknowledgments above.

Finally, I must thank those who are more officially involved with this thesis. I am very grateful for the guidance provided by my advisory committee of Justin Zobel and Steven Bird. In addition I would like to sincerely thank the two examiners of this thesis (whose identities I do not yet know) for performing the time-consuming and fairly thankless task of reading the many pages which make up this thesis and providing many useful and insightful comments which helped to shape this thesis, along with Alistair Moffat, my chair of examiners. I also thank those who taught ad hoc coursework subjects to help me satisfy the requirements of my PhD: Dan Flickinger (yet again) and Diana McCarthy.

Most importantly, I would like to thank what I consider a vital component for a successful PhD — my supervisors. My secondary supervisor Bec Dridan was incredibly helpful before she was even part of my supervision team. When, later in my PhD, she agreed to act as a supervisor (despite her imminent departure to Oslo) I found her guidance immensely valuable. My primary supervisor, Tim Baldwin, has done more than anyone else to help shape this thesis. It would be a great challenge indeed to find a better supervisor. Tim has a depth of knowledge on a wide range of research areas as well as an ability to impart that knowledge, and many connections to the broader research community. He has consistently made time to meet with me despite an ever-increasing administrative workload which seems to be the reward for being a talented researcher. He has always struck a balance between flexibility to my interests and firmness against scope creep in helping shape the direction of my thesis. I am very grateful to both Tim and Bec for agreeing to work with me.

Chapter 1

Introduction

This thesis explores *natural language processing* (NLP), which involves using software for a range of tasks where the input is natural language text written by humans. Specifically, we are interested in *syntactic parsing*, or decomposing sentences into a hierarchically nested structure from which it is possible to determine the relationship between elements of the sentence. More specifically, we explore the practical utility of a particular variety of parsing known as *deep parsing*, as well as looking for ways to maximise its accuracy over new domains.

Deep parsing characteristically produces syntactic analyses with a particularly deep structure, resolves long-distance dependencies between sentential elements and produces a deep semantic analysis as part of the parsing process (Cahill *et al.* 2008). This is often contrasted with *shallow parsing*, which directly associates a syntactic representation with a string, without, in general, an explicit semantic representation, and generally with less detail available in the analysis produced. For a example of deep parsing, assume we had parsed the following sentence:

(1.1) *John persuaded Kim to buy the fat pony which had been found by the farmer in an old quarry, eating grass.*

If a deep parser produced the correct analysis of the sentence, we would be able to infer a large number of facts from the provided semantic representation, including:

1. There are entities t, u, v, w, x, y, z
2. t is named 'John' and u is named 'Kim'
3. v is a (specific) pony
4. w is a (specific) farmer
5. x is a quarry
6. y is an amount of grass

7. v (the pony) is fat
8. u ('Kim') bought v (the pony)
9. t ('John') persuaded u ('Kim') to buy something
10. v (the pony) had been eating y (grass)
11. w (the farmer) had found v (the pony)
12. The event of w (the farmer) finding v (the pony) occurred in x (a quarry)
13. x (a quarry) was old

The semantic representations are of course more formal than we have shown here, often using propositional logic, something explicitly derived from it, or something which is similar in practice. They can represent more detailed information than is shown here, but these examples should suffice to indicate the large amount of information which is encoded in a sentence of only 21 words, and many interrelationships between those words implied by the underlying linguistic structures. We have not shown the theoretical linguistic structures here, but the truth of the above statements extracted from the sentence should be apparent to any speaker of English. Note also that many alternative sentences could produce much of the same information:

(1.2) *The farmer found the pony*

(1.3) *Kim recently bought the fat pony*

The deep analyses of these sentences would overlap with portions of (1.1), and could be related to each other, which is made possible by the deep parser abstracting away from details of the syntax such as passive or active voice (*... found the pony* compared to *... the pony was found*) and relative clauses (*... the pony which had been found* compared to *the pony was found*).

The statements we have shown were extracted from a single sentence. Consider now how repositories exist of thousands or millions of sentences. There are many more focussed stores of knowledge encoding a large amount of information as unstructured natural language. Encyclopaedias such as Wikipedia are perhaps the most obvious, but there are also collections of research papers embodying the collective knowledge of various scientific fields, question and answer forums, online news archives and many more. In all of these, most of the information is in unstructured sentences of prose written by humans (even though there may be a small amount of somewhat machine-readable metadata attached such as topic tags in many cases). Syntactic analysis, and in particular the outputs of a deep parser, can help to convert that information into a more structured format which can be more directly used computationally,

and, for example, integrated into knowledgebases or used for fine-grained retrieval of documents referring to particular events, among many other applications. Thus deep parsing is a potentially valuable tool for making information more accessible in many different ways.

Deep parsing is often performed using handcrafted *precision grammars*, which encode the allowable structures of a language using a vocabulary and a set of grammar rules specifying how sentences can be formed. The exemplar we use in this thesis is the English Resource Grammar (ERG; Flickinger (2000)), a precision grammar of English in the Head-driven Phrase Structure Grammar formalism (Pollard and Sag 1994) which can be used in a deep parsing pipeline. Until recently, many precision grammars such as the ERG have been subject to criticisms of being excessively computationally intensive to use, and of not being able to parse a sufficient proportion of sentences in real-world text. Recent developments have alleviated both of these problems, and it therefore seems as if the time has come to show how the ERG can be applied to real-world NLP tasks.

However as is often the case in NLP research, there are obstacles to realising this dream, and part of this thesis is concerned with avoiding some of these. The main problem we explore and attempt to handle is dealing with the massive ambiguity present in natural language. To get an idea where this comes from, consider:

(1.4) *The orange ducks find weeds with their beaks*

This sentence has a single obvious interpretation to a human — *ducks*, which are *orange* in colour, locate *weeds* by using *their beaks*. However, when we look deeper we see that there are many alternate interpretations.

Firstly, *with their beaks* could equally be modifying the phrase *weeds* instead of the finding event — this would be the more natural interpretation if the phrase was *with long stalks* instead of *with their beaks*. But knowing the appropriate interpretation requires real-world knowledge from the human reader — particularly that ducks have beaks and weeds do not. A computational natural language parser does not have such knowledge and must permit both possibilities.

Looking further, there is even more ambiguity present in this sentence. We instinctively interpret *orange* as an adjective[1] modifying *ducks*, but it could equally be a noun referring to the fruit. In English, it is possible to join fairly arbitrary sequences of nouns together in *noun compounds*, so *the orange ducks* could be referring to a set of aquatic birds which are associated with the fruit for some reason. This may not seem like a particularly pathological interpretation, but consider also that *ducks* can be a verb ("He **ducks** his responsibilities") and *find* can be a noun ("What a great **find**!"). This means that the sentence could be describing an *orange* (the fruit) which *ducks* a particular set of responsibilities. In this case, these responsibilities are *find*

[1]Colours can universally act as nouns as well as adjectives in English, but we ignore the former possibility for simplicity in the following discussion

weeds, which is another noun compound, referring to *weeds* which are somehow associated with a *find*. Again, *with their beaks* could modify two different items — either the *find weeds*, or the act of *ducking*. Semantically, this is completely implausible to a human reader, but these are all possibilities which must be allowed for.

And there are still more ambiguities. We have seen that *orange*, *ducks* and *find* can all be nouns, so could make up a three-word compound noun *orange ducks find*,[2] while *weeds* can be a verb ("She **weeds** the garden regularly"). This means that the sentence could also refer to a particular *orange ducks find* which *weeds*. Note also that the points of ambiguity are often multiplicative — for example, there is still uncertainty about the attachment of *with their beaks*, regardless of whether *orange* is a noun or adjective. This ambiguity comes from a sentence of only eight words. While it may (correctly) be suspected that the sentence was deliberately constructed to display a large degree of ambiguity, having a one or two instances of ambiguity is reasonably common for even short sentences. It also the case that many sentences of natural language are far longer than this, with far more opportunities for ambiguity to occur, creating exponentially more potential analyses for a sentence.

Ambiguity is in fact endemic in natural language, and dealing with this is one of the important aspects of parsing. In the context of parsing with the ERG, handling this ambiguity is a problem of *parse selection* — choosing the best parse from multiple candidate analyses of a sentence, each of which corresponds to a different set of decisions about points of ambiguity. The ERG offers 52 candidate parses for (1.4) and more than 500 for (1.1) (which was not even deliberately designed to display ambiguity), but in most processing scenarios we would only be interested in a single best analysis.

There is much prior work on methods to select the best parses, generally based on learning statistical models for how to put together parse trees from a *treebank* of ideal parse trees which has been manually created by human annotators. However a given treebank will be in a particular domain, and if this domain is not the same as the target domain of text we are trying to parse, the parsing model will generally perform suboptimally. To understand why this is, imagine that the training treebank, from which we are learning the statistical model, is drawn from an instruction manual for growing citrus fruit. We would expect many instances of *orange* in this treebank to correspond to the noun denoting the fruit, and fewer to refer to the colour, so if we gather statistics from this treebank, we would be more likely to incorrectly recognise *orange* as the fruit when parsing (1.4), leading to an incorrect analysis. As an initial step in investigating the cross-domain performance penalty, we will quantify the size of this effect for the ERG, and further to this, we will search for ways to avoid the problem if we wish to adapt to a new domain. In some cases, creating a new treebank

[2]Note that there is separate interpretation possible here, where *orange* is an adjective, which is also permissible in the compound. If the plural as the non-final element of the noun compound seems implausible, consider *parts bin*.

is the best strategy, but we will examine methods to minimise the effort required to achieve this in several ways.

In the second major part of the thesis, we move on to applications of the ERG. The first example is a biomedical information extraction system. We test the hypothesis that the deep semantic information such as long distance dependencies output by the ERG could be useful in extracting information from research abstracts in the biomedical domain — specifically, detecting modification of salient biomedical events mentioned in the prose of the abstracts. The second application chapter is concerned with the problem of matching semantic structures: while precision grammars abstract away from certain syntactic differences, there is a large class of structures with different semantic representations but underlyingly similar (albeit nonidentical) semantics according to human readers. By mining corpora of semantic structures, it should be possible to identify pairs of substructures which frequently denote similar meanings; we evaluate whether this is possible.

1.1 Research Contributions

This thesis includes a diverse range of contributions to the world of deep parsing with the ERG. We explicitly quantify the cross-domain accuracy penalty for parse selection with the ERG for the first time. In the course of this, we also perform a comparison of various well-established and newly-created evaluation metrics for parse selection, and propose a method to compare corpora numerically for syntactic similarity.

We then consider how this cross-domain loss of accuracy can be avoided. One obvious method is to create a small in-domain treebank to combine with the larger out-of-domain treebank. We show that a more informed strategy of combining these treebanks of disparate domains can lead to improved parse selection accuracy over the most obvious approach, as well as showing that the parameters for the treebank combination can be selected in advance using the small in-domain corpus.

Another case we consider is when there is some external incompatible treebank available in the domain which we are interested in parsing. We show that it is possible to transfer these superficially incompatible annotations to the ERG for learning parse selection models or to speed up treebanking. In each case the result is better parsing accuracy on the new domain.

For applications of deep parsing, we show that for an information extraction task over biomedical research abstracts, deep parsing provides useful information as input to a machine-learning approach, and evaluate the contribution of parsing accuracy in such a system, determining whether a domain-tuned parse selection model improves accuracy in a downstream task. Finally, we present a method for automatically extracting correspondence rules between distinct semantic structures from parsed corpora, and show that many of these correspondence rules are valid, in that they

can be used to create alternative semantics of unseen sentences, which can in turn be used to generate alternative semantically-similar rephrasings of the original sentences.

1.2 Structure of the Thesis

This thesis is divided into multiple parts. Part I contains background material relevant to the entire thesis. Chapter 2 is literature review of general purpose background material to assist in comprehension of the remainder of the thesis; many sections of this may already be familiar to a reader with a background in natural language processing or computational linguistics. Chapter 3 describes in some detail the specific resources used, such as grammars, corpora, and software for parsing and related activities. Most of this will be new to readers not acquainted with the specifics of the NLP tools from the DELPH-IN consortium.

Part II, which is the first part of the thesis describing novel contributions, is concerned with the problem of parse selection, which is important if we want to obtain high quality parses from our grammar of choice. In Chapter 4, we evaluate the accuracy penalty we can expect when moving across domains as well as ways to avoid it, while in Chapter 5 we look for ways to take advantage of external, superficially incompatible resources to speed up annotation or gain a free boost in parse selection accuracy.

Having laid the foundations by working on parse selection accuracy, in Part III, we look at applications — how we can actually use these deep parsing tools in NLP tasks. Chapter 6 covers information extraction over biomedical research abstracts, and evaluates the contribution a deep parser can make, while Chapter 7 is concerned with robustly matching the semantics of different sentences, and abstracting away from less significant differences for greater robustness in systems involving large scale semantic comparison. Finally, Part IV contains the conclusion chapter, Chapter 8, where we summarise the research and reflect on what has (and has not) been discovered in the course of this research, and speculate on future directions in which the research could be taken.

Part I

Background

Chapter 2

Literature Review

2.1 Introduction

In this chapter, we survey the background literature necessary to understand this thesis. This thesis falls within the computer science subfield of Natural Language Processing (NLP), although we have attempted to make this background material comprehensive enough that specific detailed knowledge of the field is not necessary — a general computer science background should be sufficient, although general knowledge of NLP or a related discipline would be helpful. As such, we provide brief introductions to some fairly standard concepts from NLP which are used heavily in this thesis, aiming to provide enough information on these to place the body of the thesis in context. In Chapter 3, we give more detail on the specific tools used in this thesis.

Sections 2.2 to 2.5 introduce concepts which are relevant background material for all chapters of this thesis, but particularly for Part II. We start in Section 2.2 with an introduction to syntax, the study of sentence structure, first using a simple account based on context-free grammars, then moving onto a brief exploration of head-driven phrase structure grammar, a more sophisticated syntactic theory which underlies all of the work in this thesis. A complete description would occupy an entire textbook; we attempt to elucidate the principles of the formalism to the extent that they influence the remainder of the thesis. Section 2.3 is a similarly brief description of machine learning. It explains only those subtasks and algorithms which are most relevant to this thesis — specifically, supervised learning using maximum entropy modelling. Machine learning approaches are used in parsing (assigning syntactic analyses to sentences), which we explore in Section 2.4, along with the various factors which make this difficult. We also outline some approaches different researchers have taken to parsing, contrasting those which are used in this thesis with the approaches used elsewhere. Syntactic parsing also requires supporting tools, which we explore in Section 2.5.

The remainder of this chapter gives background knowledge which is particularly relevant for certain specific chapters of the thesis (although there is often overlap). Section 2.6 talks about approaches to porting a parser to different domains (styles of text), which is particularly relevant for Chapters 4 and 5, while the discussion in Section 2.7 on mapping between superficially incompatible syntactic representations is mostly relevant for Chapter 5. The remaining section is useful background material for the more application-focussed Part III of the thesis. Section 2.8 covers in some detail a task relating to extracting information from biomedical research abstracts which is the basis of Chapter 6.

2.2 The Syntax of Natural Language

2.2.1 Introduction

In this section, we give a very brief introduction to the very broad field of natural language syntax, or the analysis of sentences. In particular, we aim to give some detail on a theory of syntax known as Head-drive Phrase Structure Grammar, or HPSG, which is used extensively in this thesis. To explain this theory, in Section 2.2.3 we follow a fairly common approach of introducing syntax using context-free grammars, a method of formally specifying languages which can be applied to natural language. In doing so, we will also note some of the limitations of such an approach, which theories such as HPSG are designed to address. There may be an inevitable bias towards exploring phenomena which HPSG is particularly good at handling.

We use English for the linguistic examples in this section (and indeed throughout the thesis), as this is the language used in subsequent work in this thesis, and is also familiar to readers of the thesis. We attempt to highlight where there is a particularly strong bias towards English, although almost none of the linguistic structures we show are completely language-independent.

2.2.2 Overview

The study of natural language syntax seeks to explain, in part, the set of sentences which constitute a particular human language. There is an infinite variety of novel sentences which are valid in a language and which can be understood by other competent speakers of the language. From this we can infer that the meaning of the sentence is somehow encoded in the sentence components and how they are combined within the sentence structure. Native speakers of a language can understand and produce an effectively infinite variety of sentences. However, these sentences are still strongly constrained; there is also an infinite variety of sentences (some of which are super-

ficially similar) which would be considered ill-formed by speakers of the language,[1] and should thus not be considered as part of the language.

Syntactic theories are concerned with explaining these phenomena: why some sequence of sounds or written characters (in this thesis we only look at the latter) should be a valid sentence encoding meaning, while some other sequence would be considered by speakers of the language to not be part of the language, and may not convey any meaning at all.

The sentences which are considered part of a language are denoted *grammatical*, since they conform to a hypothetical grammar underlying the language, while strings of words which are not well-formed are *ungrammatical*, and conventionally prefixed with a '*' to indicate this.

For example, speakers of English would generally consider (2.1) grammatical, but not (2.2) and (2.3).

(2.1) *The fat pony sleeps in the barn*

(2.2) **Barn sleeps fat the pony in*

(2.3) **The pony barn the the*

A theoretical account of a language will in general be designed to mirror the judgments of a native speaker. That is, it will seek to permit all grammatical sentences while excluding all ungrammatical ones, so that we can tell whether any arbitrary string is a sentence of the language. If a grammar excludes grammatical sentences, it is said to *undergenerate*, while if it allows ungrammatical sentences, it is said to *overgenerate*. The terminology here comes from the interpretation of syntactic analysis as *generating* sentences by applying some set of grammatical rules.

A grammar should avoid both undergeneration and overgeneration. However, there is an obvious tension between these two goals. It is trivial to create a grammar of a language which accepts any string and thus correctly permits all grammatical sentences. Conversely, one can easily create a grammar which allows no sentences, thus correctly rejecting ungrammatical sentences. However, neither of these theories is very informative about the underlying nature of the language in question for two reasons. They fail to create meaningful boundaries around the set of allowable sentences, and they are also unlikely to be useful in interpreting a sentence of the language. A well-constructed account of a language seeks to differentiate accurately between grammatical and ungrammatical sentences, and provide, to some level, an explanation for this differentiation.

[1] They may not even be meaningful, although as Chomsky (1957) famously pointed out, syntactic well-formedness does not directly correspond with conveying meaning

2.2.3 Context-Free Phrase Structure Grammars

Definition

Formal language theory (Salomaa 1973) is concerned with the strict mathematical definitions of a language – that is, a (possibly infinite) set of strings composed of some vocabulary. It is concerned with representing such infinite languages using a finite specification. There are various formally-defined methods of creating these specifications, which can be arranged in a hierarchy of expressiveness. *Context-free languages* have a level of expressiveness which falls between *regular languages* and *context-sensitive languages*. The question of whether context-free languages are sufficiently expressive to encode all human language has proven to be a controversial one, which we discuss further at the end of the section. For now, it is useful for expository purposes to demonstrate the modelling of a natural language as a context-free language, since we can successfully explain many phenomena using this level of expressivity.

A context-free grammar, or CFG, is a way of encoding a context-free language. It is a special case of a *generative grammar* (Salomaa 1973) in the technical sense, which consists of a set of *terminals*, a set of *non-terminals*, a starting symbol drawn from the non-terminals, and a set of *productions*. Terminals are symbols which can not be further expanded, and can thus terminate the sequence of rule applications. Each production, which is of the form $P \rightarrow Q$, indicates that the left-hand side (P) can be rewritten as the right-hand side (Q). Q is a sequence of terminals and non-terminals while P, in a context-free grammar, must be composed of exactly one non-terminal, so that left-hand side of the rule is not conditioned on any context, making it *context-free*.

Applying CFGs to Natural Language

CFG accounts of language are relatively simplistic, and as such there are some associated problems which we explore below. However, the concepts used by CFGs are relevant for many other syntactic theories and NLP resources.

To apply CFGs to natural language, we generally divide sentences into *tokens*, which often correspond to what are generally thought of as the words of the sentence. These form the alphabet (repertoire of terminal symbols) of the CFG. There is generally a notion of *part-of-speech*, or POS, associated with the tokens, which is something like the traditional word classes such as *noun*, *verb*, *adjective*, *preposition* and *article*. These parts of speech are pre-terminals, sitting at a level higher than the token.

The components of a sentence are arranged in a tree structure, where each node of the tree could correspond to some kind of phrasal category, such as *noun phrase* or *verb phrase*, some intermediate level category (such as a part-of-speech) or a token. These intermediate structures are considered to be *constituents*, which are units of

one or more word tokens, corresponding more-or-less with internal boundaries as they may be postulated by a speaker of the language, and are generally meaningful in the sense that they can be interpreted (to some degree) by a speaker of the language. There are also various linguistic tests for constituency, such as the ability to conjoin constituents of matching types to each other (Chomsky 1957: Chapter 5). We can see a demonstration of this if we consider the sentence *The pony in the barn saw a duck*. *The pony*, *in the barn* and *saw a duck* could all be considered as meaningful, and form answers to questions, such as *Who saw the duck?* or *What was it the pony did?*. We can also form conjoined phrases using these (*The pony and the cow*, *in the barn and behind the farmer* or *saw a duck and fainted* respectively). In contrast, subsequences such as *The pony in*, *in the barn saw* and *pony in the* do not seem to form a natural syntactic or semantic unit, so would not be considered constituents. Note that it is possible for constituents to be nested within other constituents – for example, the constituent *the barn* is a subsequence of *in the barn* and *The pony in the barn*, which are both also constituents.

To create a CFG for a natural language, we create productions corresponding to the phrasal patterns that we believe match the constituents in the language. This CFG can be used to generate sentences of the context-free language, which we hope corresponds with the natural language we are attempting to model. By recording these rule applications, we obtain a hierarchical tree structure for the sentence. That is, the productions in a CFG are specified so that the left hand side of each production corresponds to a higher level node in the tree and the right hand side corresponds to its child or children. This tree is known as a *phrase structure tree*, and the productions can also be denoted *phrase structure rules*.

For small sets of grammatical and ungrammatical examples, it is relatively easy to come up with a context-free grammar which accounts for the observed phenomena, and we work through an example of this here. We first examine the phrase *The fat pony* from (2.1), consisting of the noun *pony*, the adjective *fat*, and *The*, which we consider a type of determiner along with other similar items such as *this* and *those*. We would generally categorise this whole phrase as a *noun phrase* since it is fundamentally focussed on the noun. It would make little sense if the noun was absent, and we can replace it with a different noun-like object – specifically a pronoun like *it*. The adjective *fat*, on the other hand, can be freely omitted while still leaving a well-formed phrase. Similarly, *the barn* from the example would also be considered a noun phrase.

Additionally, *sleeps in the barn* is a *verb phrase* as it relies heavily on the verb, and we can replace it with another verb-like object such as *does so*. Finally, we classify *in the barn* as a *prepositional phrase*. It is introduced by the preposition *in*, and in at least some contexts this phrase can be replaced by a bare preposition: *He put the pony in the barn* versus *He put the pony in*.

These noun phrases, verb phrases and prepositional phrases generally contain other phrases, leading to the aforementioned tree structure of the sentence. We ab-

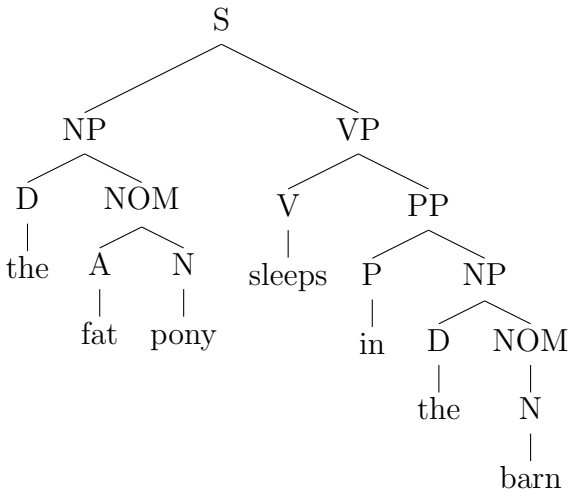


Figure 2.1: A phrase structure tree for (2.1)

breviate noun as 'N', adjective as 'A', verb as 'V', preposition as 'P' and determiner as 'D'. These parts-of-speech correspond somewhat with traditional word classes, which is not coincidental. They allow us to group together words with similar syntactic functions, acting something like variables, and enabling generalisation from a small number of rules as well as the creation of a relatively compact grammar. The corresponding phrasal types are shown as the part-of-speech abbreviation followed by 'P' for phrase, so that 'NP' denotes 'noun phrase', and so on, while we denote the entire sentence as 'S'. We also have a category 'NOM', which is an abbreviation of 'nominal'. This occurs at a level below NP, before the determiner is attached. The linguistic justification for such a category is sound, although complex for this brief example, but we use it primarily so that we can create an analogous structure in Section 2.2.4.

We can then draw the phrase structure tree for the sentence shown in Figure 2.1. This tree actually embodies many assumptions about linguistic theory (such as having two levels of hierarchy in the noun phrase) but we will accept its structure without question for the time being. A CFG enables us to construct such trees, and thereby evaluate a string for grammaticality, as we shall see below.

Let us assume for the moment that the subset of the language that we wish to cover includes the sentences in (2.4) to (2.8).

(2.4) *The fat pony sleeps in the barn*

(2.5) *This tiny pony sleeps*

(2.6) *The pony under the tree admires this big duck*

S → NP VP
NP → D NOM
NOM → A* N (PP)
VP → V (NP) (PP)
PP → P NP
V → ‘sleeps’ | ‘admires’
N → ‘barn’ | ‘duck’ | ‘pony’ | ‘tree’
P → ‘in’ | ‘under’
D → ‘the’ | ‘this’

Figure 2.2: A grammar to account for (2.4)– (2.8)

(2.7) *The duck in the barn sleeps*

(2.8) *This big fat duck admires the tiny barn*

Using the established set of labels, we could encode the grammar and lexicon to account for these examples as shown in Figure 2.2. For notation, we use parentheses to indicate optionality, ‘*’ to indicate zero or more repetitions, and ‘|’ to separate alternatives for the right-hand side of a production (The use of ‘*’ gives this grammar an infinite number of rules, which technically means that it is not a CFG, however it is a convenient shorthand to use for expository purposes and it has little effect on the following discussion)

Determining whether the grammar permits a particular sentence then becomes a matter of determining whether there is some set of productions which can produce the sentence.[2] For this small grammar, we can do this manually, although as grammars become larger and more complex, this becomes unfeasible. We will examine methods for applying this on a larger scale in Section 2.4.

For the very small artificial subset of English we are attempting to explain, this grammar seems to do a reasonable job. It correctly permits the target sentences and correctly excludes other examples such as those in examples (2.9) to (2.11), as well as the earlier ungrammatical examples (2.2) and (2.3).

(2.9) **Fat admires pony the*

(2.10) **Pony sleeps the barn in*

[2]Note that this CFG, when used in a linguistic context, is not simply a recogniser for valid sentences of a language — there is also linguistic significance in the history of applying the productions, since this determines the shape of the phrase structure trees.

(2.11) **Duck admires tiny barn*

The grammar has a number of interesting properties which are worth exploring. Firstly, it permits a language of infinite size, which is desirable, as it is a necessary condition for modelling something with infinitely many possible sentences such as a human language. To see how this works, note that a NOM can contain a PP, while a PP requires an NP, which in turn contains a NOM, thus permitting another embedded PP, meaning that we can have infinitely many levels of nesting within an NP. In (2.12), we show an example of five levels of nesting, although the semantics of the phrases may seem strange due to the highly constrained vocabulary.

(2.12) *The duck under the pony under the duck in the barn under the tree under the pony....*

This example could continue on indefinitely in the language specified by the CFG, although it is likely to be already reaching the limits of computability for a human. Nonetheless, recursion in its various forms is an important part of natural language, and can be reproduced by even a very small grammar.

The second interesting feature of the grammar is that, despite its small size and the constrained language it is modelling, it still permits ambiguity. Consider the sentence in (2.13):

(2.13) *The duck admires the pony in the barn.*

The PP *in the barn* could be daughter of a NOM node and sister to the N node above *pony*, forming part of the noun phrase *the pony in the barn*. This corresponds to the interpretation where the pony in question is physically located in the barn, and could be paraphrased as (2.14). Alternatively, the PP could be the child of a VP, with the V node above *admires* and the NP *the pony* as its sisters, corresponding to the interpretation where the barn is the location for the act of admiration itself, as roughly paraphrased in (2.15).

(2.14) *The duck admires the pony which is in the barn.*

(2.15) *In the barn, the duck admires the pony.*

This is an example of PP-attachment ambiguity which is a pervasive phenomenon in English (Ratnaparkhi *et al.* 1994; Baldwin *et al.* 2009). This is one of many kinds of *structural ambiguity* present in natural language where a decision about the sentence structure cannot be made on syntactic grounds alone. In most cases, the ambiguity is not a barrier to interpretation by humans — indeed, humans are not consciously aware of most examples of ambiguity, since the implicit external knowledge which humans use in interpreting language strongly pushes us towards one interpretation. In this case, most people would have such a strong preference for the interpretation in

(2.14) that the other would not even be considered. However, a computational model of the language must permit both versions of PP attachment — under NOMs and under VPs — as these are both acceptable locations, even when one interpretation is more semantically plausible to a human. This grammar is therefore inherently ambiguous, and this must be the case if it is to accurately model a human language. The problem of ambiguity in grammars modelling natural language is one which is frequently encountered, and the size of the problem increases as sentence lengths and grammar sizes increase. Any system which seeks to assign a unique interpretation to a given sentence must find some way of handling this. We will return to this problem in Section 2.4.1

Problems with the CFG Account

One obvious question when we are attempting to model natural languages using a CFG is whether all natural languages are indeed capable of being modelled with the expressive power of a CFG — that is, whether natural languages are context-free, which turns out to be difficult to answer. Chomsky (1957) and others have argued that they are not, but Pullum and Gazdar (1982) showed that these putative examples previously published could indeed be handled as context-free languages.[3] However Shieber (1985) presents a counterexample from Swiss German which is provably non-context-free.

In many ways the answer to this question in terms of formal language is irrelevant in terms of deciding whether to use an unadorned CFG to model a given language. Whether or not it is possible in terms of expressive power, there are many reasons unrelated to formal language theory which are arguments against using bare CFGs — in particular, it is important for us that the accounts *are* linguistically satisfactory. The small grammar of the Figure 2.2 was reasonably successful at explaining the artificial subset of English we carefully chose. However, when we attempt to explain the much wider range of syntactic phenomena present in all natural languages, we need to fall back on some options which seem linguistically suboptimal at best. There are many such phenomena we could highlight, but we will only cover two very simple examples.

The first is *agreement* (Pollard and Sag 1994: Chapter 2). In English and many other languages, this is a pervasive phenomenon, where particular elements in a sentence are required to take different forms due to other syntactically-related items in order to *agree* with each other for certain grammatical features, such as number (i.e. plurality) or gender. This occurs despite the fact that the semantics of the relevant feature could be determined from one of the words only.

[3] Although they note that the analyses required to achieve this are often linguistically suboptimal, and that linguists frequently conflate the tasks of creating a linguistically satisfactory account of a language and merely accounting the allowable strings in that language.

We wish to explain why examples (2.16) and (2.17) are grammatical while (2.18) and (2.19) are not:

(2.16) *The ponies sleep*

(2.17) *The ducks admire the ponies*

(2.18) **The ponies sleeps*

(2.19) **The duck admire the ponies*

The reason of course is that in selecting a verb form to use, the speaker must pay attention to the number (singular or plural) of the noun which is the *subject* of the verb. This is an example of *subject-verb agreement*, which is particularly prevalent in English. Another form of agreement is between particular determiners such as *this* and *these* which must agree in number with the corresponding nouns, as indicated by the ungrammatical examples in (2.20) and (2.21).

(2.20) **This ponies admire the barn*

(2.21) **These pony sleeps*

We can of course explain these in the framework we have already established, by splitting the nouns and verbs into different singular and plural classes, and creating corresponding singular and plural variants of the NOM, NP, VP and S rules. We show an implementation of this in Figure 2.3. This has solved the problem of explaining these few agreement examples but is far from ideal, as it has created a profusion of new categories and grammatical rules (especially considering each rule variant separated by '|' corresponds to a new rule) and much redundancy.

A related problem is that of *subcategorisation* (Gazdar 1985: Section 2.5): verbs have certain restrictions on the arguments they take. All permit a subject, but some verbs require a noun argument in the direct object slot (which in English generally immediately follows the verb), which are known as *transitive* verbs, and include the verb *admires* from (2.8), where *the tiny barn* is the direct object. Other verbs such as *sleep* are *intransitive*, and a direct object is not permitted. So, a satisfactory account of English should be able to account for the ungrammaticality of (2.22) and (2.23):

(2.22) **The pony sleeps the barn*

(2.23) **The tiny duck admires*

S → NP_sg VP_sg | NP_pl VP_pl
NP_sg → D NOM_sg | D_sg NOM_sg
NP_pl → D NOM_pl | D_pl NOM_pl
NOM_sg → A* N_sg (PP)
NOM_pl → A* N_pl (PP)
VP_sg → V_sg (NP_pl) (PP) | V_sg (NP_sg) (PP)
VP_pl → V_pl (NP_pl) (PP) | V_pl (NP_sg) (PP)
PP → P NP_sg | P NP_pl
V_sg → ‘sleeps’ | ‘admires’
V_pl → ‘sleep’ | ‘admire’
N_sg → ‘barn’ | ‘duck’ | ‘pony’ | ‘tree’
N_pl → ‘barns’ | ‘ducks’ | ‘ponies’ | ‘trees’
P → ‘in’ | ‘under’
D → ‘the’
D_sg → ‘this’
D_pl → ‘these’

Figure 2.3: A grammar to account for (2.4)–(2.8) and (2.16)–(2.21)

Similar to agreement, we could also handle this by the creation of new categories. We could remove the broad catch-all verb category, and instead assign the verbs to new categories corresponding to transitive and intransitive verbs, and create two variants of the VP production for each verb class, which would account for the above examples. Again, this is creating a profusion in the number of labels in our grammar, and the situation becomes even more complex when we consider the other possible subcategorisations: verbs can also be optionally transitive (like *eat*) or permit two arguments (like *give*), so the number of required categories would expand even further.

For both subcategorisation and agreement, we have seen solutions which multiply the number of categories and productions. This is not necessarily a problem in itself — we should expect that a grammar handling something as complex as a human language should itself be somewhat complex. But if we need to multiply the size of the grammar by a factor of two or more to account for every linguistic phenomenon, the size of the grammar will increase exponentially and rapidly become unmanageable.

However this is not the only reason the ‘make new categories’ solution is suboptimal. The goal of a grammar is not only to permit and exclude the correct sentences — there are a number of other important considerations. Firstly we want it to be linguistically reasonable. The resulting phrase structure trees and the grammar itself should be plausible to a linguist. One aspect of this is that we would like the commonalities between linguistically similar items to be apparent. The solution we

proposed above fails in this regard. In a CFG, there is no notion of relatedness apart from exact matches between the arbitrarily-assigned labels. There is no grammar-internal notion of similarity between the 'N_sg' and 'N_pl' categories in Figure 2.3,[4] even though from a linguistic perspective, they share much and differ only in one small respect. The same would be true for the hypothetical new categories for transitive and intransitive verbs. Similarly, we have no explicit encoding of the fact that 'N_sg' and 'V_sg' mean that the subject of the verb is singular (and that there is no such correspondence with the object of the verb). Additionally, because of this duplication of categories, we have a large amount of repeated information. As there is no notion of commonality between, for instance, transitive verbs and intransitive verbs, we require two very similar verb phrase rules to explain linguistic phenomena which share a lot in common.

Leaving aside the handling of subcategorisation and agreement, there is another consequence of the arbitrariness of CFG labels. There is no explicit relationship between certain categories which seem like they should be related. For example, nouns seem to play a crucial role in noun phrases, being mandatory and affecting characteristics such as number. However, in the set of productions we have shown above, 'N' and 'NP' are simply arbitrary labels, and there is no explicit connection between them. Similar effects apply for verbs in verb phrases and prepositions in prepositional phrases. Linguistically this is referred to as *headedness*: phrases generally have a particular privileged item known as the *head* which determines many properties of the overall phrase. For noun-phrases this is, unsurprisingly, the noun, for verb-phrases this is the verb, and so on. Ideally we would like our formalism to allow the relation between these heads and their phrases to be made clear, but this is not possible with a CFG in its unmodified form. It is important to note that none of these problems are related to the expressive power of the CFG — the solutions we have presented are perfectly acceptable in terms of formal language theory, but they simply do not give analyses which are satisfactory in linguistic terms.

2.2.4 Head-driven Phrase Structure Grammar

To account for these deficiencies with CFG-based accounts of language noted in Section 2.2.3, linguists have proposed a range of solutions with more elegant ways to explain a large range of phenomena, all of which have different strengths and weaknesses. In this section we give a brief summary of one such formalism designed to provide a linguistically-motivated account of a range of phenomena, which is also the basis of subsequent NLP techniques used throughout this thesis. This formalism is known as Head-driven Phrase Structure Grammar (Pollard and Sag 1994; Sag *et al.* 2003), or HPSG. The summary in this section is largely based on Sag *et al.*

[4]The labels have mnemonic value so the common prefix might indicate commonality between these to a human, but this is nothing to do with the CFG itself.

(2003: Chapter 3), although we indicate where there are divergences. The examples we show of the principles, such as demonstration grammars, are also based on this work, although we are attempting to explain a different set of expository phenomena, so there are noticeable deviations as well.

Note that not all of the ways of solving linguistic problems in this section are intrinsic to HPSG. Many aspects, such as choice of feature values, are more appropriately viewed as 'implementation details', although they are generally closely related to how the problem would be solved in HPSG-based grammars of natural languages, and in particular English, the language of choice for the worked examples. This section does not contain a complete treatment of HPSG; the reader is referred to the canonical reference (Pollard and Sag 1994) and textbook (Sag *et al.* 2003) for further information.

Encoding Information in the Lexicon

CFGs as discussed in Section 2.2.3 take what could be considered a "top-down" approach to defining a language, specifying categories (such as parts-of-speech) and members of those categories. This approach is attractive for its simplicity, although we touched on some of the shortcomings of this approach when we looked at how different lexical items have different behaviours. For example, handling differences in verb transitivity requires dividing up the verb category because some verbs expect different arguments, and this behaviour is lexically-conditioned.

One of the important innovations of theories such as HPSG is to take a more "bottom-up" approach, so that much of the important information in a grammar of a particular language is encoded in the lexicon,[5] as noted in Sag *et al.* (2003:Chapter 8). A detailed lexicon allows us to account for differences in behaviour of lexical items, such as their expected complements, on the lexical entries themselves. The rules of a grammar are constructed to make use of these lexical specifications. Generally, inheritance is used to avoid duplication of information so that we do not need to repeatedly specify the pattern of arguments allowed by a strictly intransitive verb, for example.

Feature Structures

HPSG is one of several syntactic formalisms based on *feature structures*.[6] Very loosely, feature structures can be thought of as augmentations of the nodes of a

[5] HPSG is by no means unique in this regard. Steedman (2000) Bresnan (2000) and Joshi and Schabes (1997) all describe alternative formalisms where much of the information is encoded in the lexicon.

[6] Many other formalisms also use feature matrices to encode syntactico-semantic information. Chomsky (1965) presented the use of (very simple) features, while LFG uses feature-based representations heavily (Bresnan 2000). There are also substantial differences in each case from HPSG in how they are used.

CFG-like representation with salient properties of the node. They are used to record the information associated with units of language — in particular, there is a feature structure associated with each unit of written or spoken language, such as a word, phrase or sentence. There is also a feature structure for each lexical entry — the abstract representation of the meaning associated with a particular lexical item which can be realised as a word in a sentence. For this discussion we treat lexical entries as identical to the corresponding words, although this is an oversimplification.

A feature structure is generally represented in an *attribute-value matrix* (AVM), which is a notational convention for displaying the pairs of features and values. For example, in one formulation, we might have the following very simple lexical entry for the noun *cat* (this is much simplified from what we would find in most HPSG implementations):

(2.24)
$$\begin{bmatrix} \text{POS} & \textit{noun} \\ \text{NUM} & \textit{sg} \\ \text{PERS} & \textit{3} \end{bmatrix}$$

From this, we can see that the part-of-speech (POS) of *cat* is *noun*, the number (NUM) is *sg*, or singular, as opposed to plural, and the person (PERS) is *3*, indicating that the word denotes a third-person reference.[7] Note that we would probably not see this exact feature layout or set of feature names in a fully-fledged HPSG account of a language, but the general principles apply.

By using these feature values to encode relevant properties, we can go some way towards using HPSG to solve some problems of CFG. The solution we mentioned for handling agreement with CFG was to divide up nouns into separate categories, but noted that this doesn't reflect what nouns have in common. By encoding number simply as a property, we treat singular and plural variants as elements of the same noun category with a different value for that property.

However, the feature structures in HPSG are actually more sophisticated than shown in example (2.24). The values can be nested, so that instead of the values being atomic, they can be other feature structures. Returning to the above example, we might notice that both person and number are important for explaining linguistic phenomena such as verbs inflecting for agreement, which we already mentioned in Section 2.2.3. We noted that for a regular English verb such as *sleep*, there is a difference between third-person singular (*she sleeps* or *the pony sleeps*) and third person plural (*they sleep*). However, the situation is a little more complex than this. The verb inflection for third person plural is also used for first person singular or plural (*I run*/*we run*) and second person (*you run*). On this basis, we may decide that it is sensible to group NUM and PERS into a single feature called 'agreement', or AGR to give us a new feature structure for *cat*:

[7] All common nouns are third-person, but this is not the case for some personal pronouns such as *you* which refers to the addressee (i.e. second person) rather than some third party.

$$(2.25)\quad \begin{bmatrix} \text{POS} & \textit{noun} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \textit{sg} \\ \text{PERS} & \textit{3} \end{bmatrix} \end{bmatrix}$$

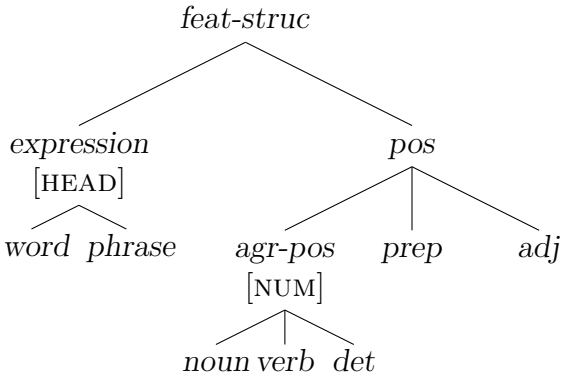
As a notational convention, it is also customary to refer to paths through these complex feature structures by listing the applicable feature names from the outside inwards, separated by '|', so that instead of referring to the value of the NUM feature of AGR, we could refer to AGR|NUM. This notation can also be used within feature structures.

Type Hierarchies

Another important characteristic of these feature structures is that they are *typed* — that is, every feature structure and value has a type associated with it, and these types are arranged in a hierarchy, with feature values inherited from the parent (or parents) in the hierarchy, and augmented with the features and values associated with the leaf type. The values in a feature structure like the one shown above would themselves be feature structures from the hierarchy, rather than, for example, string values as it may appear from the notation above. The combination of inheritance and type constraints (as described later in this section) means that the values for these features can be constrained to sensible values. For example, it does not make sense to have a value for POS of *sg*, and HPSG provides ways to enforce this.

The type hierarchy is designed to reflect the commonality between the various types, avoiding duplication as much as possible. It can therefore be used to explicitly indicate the commonality between closely related items, such as singular and plural nouns, or transitive and intransitive verbs. They would inherit from a common parent which specifies features shared by all subtypes, with differences encoded on the lower level types. Introducing features on the higher level types also allows us to ensure that entities only include features which are linguistically sensible. For example, NUM makes sense for nouns, but not for prepositions. This notion of typing also helps to address some problems with CFGs mentioned in Section 2.2.3 — that information is duplicated and commonality between closely related items is not apparent, since the hierarchy allows this information to be specified in a single location and inherited by subtypes.

For an example, we show the basis of a very simple type hierarchy in Figure 2.4 (the model assumed here is incompatible with the expository example (2.25)). In the figure, the words 'HEAD' and 'NUM' in square brackets below the type names refer to feature names, and indicate that these particular features are relevant for these types and their descendants in the hierarchy, since the features are inherited from the parents. Further to the type hierarchy, our grammar includes constraints on the allowed values of NUM and HEAD, as shown in the table. Later we will see how we use



Feature	Value type
HEAD	*pos*
NUM	{ *sg*, *pl* }

Figure 2.4: A sample feature structure hierarchy and feature value type constraints for a small grammar of English. *sg* and *pl* are atomic values, and the braces indicate a set of allowed values

these features in conjunction with the rules to make the grammar enforce appropriate agreement constraints.

The *pos* type refers, as we might expect, to parts of speech (which are also values of properties within the feature structures of *word*s, as we will see later). The five parts-of-speech we consider here should be fairly familiar from Section 2.2.3: there are *noun*s, *verb*s, *adj(ective)*s, *det(erminer)*s and *prep(osition)*s. There is also a further subdivision here: *noun*s, *verb*s and *det*s all fall under the *agr-pos* category, to reflect the fact that the wordforms which have these POSs are subject to agreement phenomena, conditioned by the number (singular or plural) of the corresponding noun, as we discuss in more detail later in this section. In contrast, *prep* and *adj* show no variance in these conditions for English.

The second-level *expression* type would be used, in broad terms, to refer to concrete instantiated sentence elements. For example, a sentence such as *the pony sleeps* might be analysed as shown in Figure 2.5, where we have used similar labels to those in Section 2.2.3: 'NP' for noun phrase, 'VP' for verb phrase, 'D' for determiners such as *the* and so on.

In this example, the leaf nodes *the*, *pony* and *sleeps* would be considered *word* instances, while the higher-level nodes 'S', 'NP', 'NOM' and 'VP' are instances of the *phrase* type. These are grouped together under the *expression* subtype since there is much in common between them compared to the more abstract grammatical categories elsewhere in the tree.

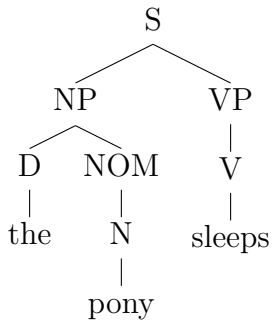


Figure 2.5: Analysis for *the pony sleeps.*

The NUM feature which is introduced by the *agr-pos* type is used to describe the relevant property of number, which is important for agreement in cases such as those shown in (2.16) and (2.17).[8]

Another feature which appears in the type hierarchy is the HEAD feature, which is introduced on the expression type, and is used to handle the notion of *headedness* discussed in Section 2.2.3 — each *word* and *phrase* instance has a head associated with it. So, for example, a noun phrase would be a *phrase* instance with *noun* as the value of its HEAD feature. These head features also apply to *word*s, so that the feature structure corresponding to an individual token like *pony* would also have a *noun* as its HEAD value, while *the* would have *det*. In fact, these head values are propagated between the various *expression* feature structures in the sentences, using the rules discussed in the following subsection.

We can show a small example of type inheritance using this hierarchy. The type *noun* inherits from *agr-pos*, and from the hierarchy we know this includes NUM as a salient feature. From the rules of inheritance, this means that the feature structure in (2.26) is well-formed.

$$(2.26)\quad \begin{bmatrix} \textit{noun} & \\ \text{NUM} & \textit{pl} \end{bmatrix}$$

Having introduced the notion of headedness, we can now specify the underlying feature structures of some of the CFG-inspired node labels as used in Figure 2.5. 'NP'

[8]For simplicity, we are ignoring here the other important factor in agreement in English of *person*, which can be first, second or third. This is needed to explain why *I sleep*, with a first person singular subject, has a different verb form to *He sleeps*, with a third person singular subject. In practice, this would generally be handled by using an AGR feature on *agr-pos* types, with NUM and PERS as subfeatures, but we ignore this possibility here to avoid introducing unnecessary complexity into the example.

corresponds to the noun phrase definition previously discussed: a *phrase* headed by a *noun*. Similarly 'V' corresponds to a *word* headed by a *verb*. Both of these definitions are shown in AVM notation in (2.27) and (2.28), where the line at the top of each AVM denotes its type (following reasonably standard conventions).

$$(2.27)\ \text{NP} = \begin{bmatrix} \textit{phrase} & \\ \textsc{head} & \textit{noun} \end{bmatrix}$$

$$(2.28)\ \text{V} = \begin{bmatrix} \textit{word} & \\ \textsc{head} & \textit{verb} \end{bmatrix}$$

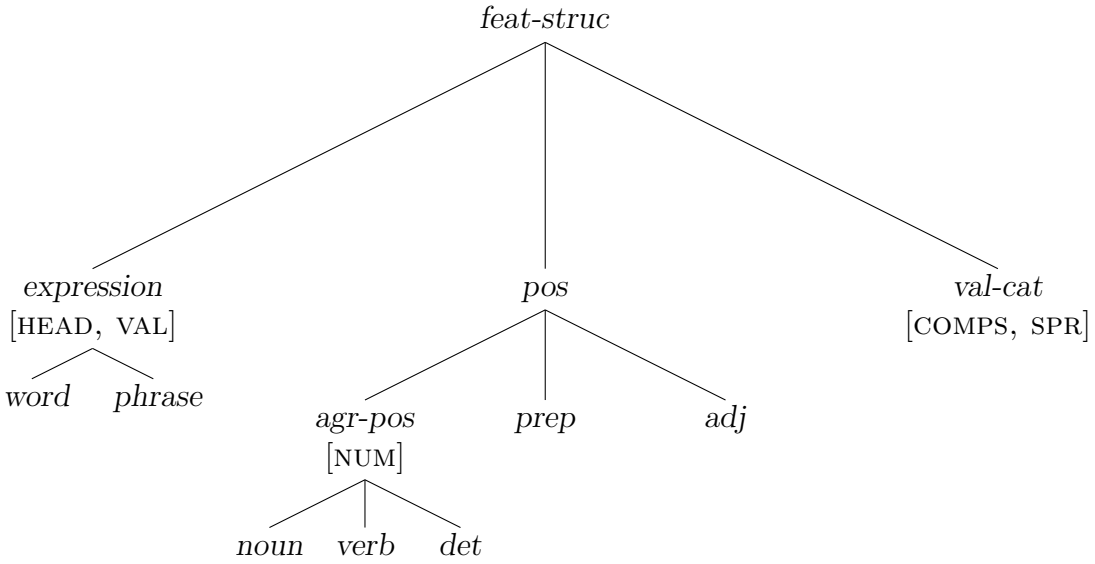
We can also create similar definitions for other POS/*expression* combinations (however we do not yet have a way to distinguish 'NOM', the label for nominals, from NPs). Note that in this view there is nothing special about these node labels — they are simply a human-readable shorthand notation for nodes conforming with particular features and values, although the labels are generally chosen for compatibility with somewhat similar structures from CFG analyses.

These feature structures for the categories are examples of *underspecification*, in that many features we would expect on instantiated words and phrases (such as NUM) do not occur here. This means that the descriptions in (2.27) and (2.28) match a larger range of feature structures than they would if they were fully specified, since they are noncommittal about the values of the omitted features. It also allows us to create linguistically sensible groupings, so that verb instances can be grouped together regardless of the value of, for example, the NUM feature. Using a CFG we had no notion of the relatedness of such very similar categories if we wished to satisfactorily handle these agreement phenomena.

Handling Syntactic Phenomena using Rules

These feature structures are integrated into grammar productions which are somewhat parallel to those in the CFG of Figure 2.2, but using feature structures instead of the simple non-terminal symbols of CFGs. The output of each production is also a feature structure, so that when we analyse a sentence using HPSG, we determine whether it is grammatical, and also produce a TFS for the whole sentence. This is useful for a range of purposes including producing a semantic output, which we discuss later, in Section 3.5.4.

Before showing how these rules work, however, some additions to the type hierarchy are needed, to add features that encode the handful of syntactic phenomena we have so far discussed. Firstly, we add a type *val-cat* into the hierarchy, immediately inheriting from *feat-struc*. This type introduces two new features: COMPS and SPR. We also add a new feature VAL (for 'valence') to the *expression* type, and stipulate



Feature	**Value type**
HEAD	*pos*
VAL	*val-cat*
NUM	{ *sg*, *pl* }
COMPS	{ *itr*, *tr* }
SPR	{ –, + }

Figure 2.6: A more extended feature structure hierarchy and set of type constraints for the feature values, for a small grammar of English, building on the one shown in Figure 2.4

that the value of this feature is an instance of the *val-cat* type. The new hierarchy is shown in Figure 2.6.

The COMPS feature name is an abbreviation for 'complements'. It is used to handle compulsory arguments[9] such as *in* ***the barn*** or *admires* ***the pony*** which are required by certain *expression* instances; in English, they generally occur after the word for which they are arguments.[10] In particular, this feature can handle different forms of verb transitivity, as transitive verbs require exactly one NP complement, while intransitive verbs require exactly zero such complements.[11] COMPS can in fact be viewed as a generalisation of verb objects. The COMPS feature takes on values of *itr* for intransitive (i.e. no complements needed or permitted) or *tr* for transitive (which for our purposes means exactly one complement). The notion of intransitivity is also extended to phrases, and other items which do not permit complements, such as NPs and adjectives, so these will also be specified as [COMPS *itr*].

The SPR feature, whose name is an abbreviation of 'specifier', generalises the notion of determiners which we see on noun phrases. Instances of *expression* which are [SPR $-$] have a 'gap' in their SPR slot and require a specifier on the left — this would be the case for expressions such as the 'NOM' node in Figure 2.5. Meanwhile those that are [SPR $+$] already have this slot filled (as would be the case for NP instances, which already have the determiner) or simply never needed a specifier at all (which would be the case for pronouns such as *you* or proper nouns such as *Kim*). But we also generalise this to VPs, treating the subject NP as the specifier of the corresponding VP, so that a sentence is simply a VP with the specifier slot occupied by the "specifying" subject NP.

We show in (2.29) how we could construct one simple grammatical rule, to create verb phrases from intransitive verbs, as the HPSG analog of the CFG rule 'VP → V', although not the versions which allow NPs and PPs.[12]

$$(2.29)\quad \begin{bmatrix} \textit{phrase} & \\ \text{HEAD} & \boxed{1} \\ \text{VAL} & \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{word} & \\ \text{HEAD} & \boxed{1} \\ \text{VAL} & \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix}$$

[9] More sophisticated analyses allow optional arguments as well.

[10] Since the subject of the verb is compulsory, it may seem logical to treat this as a complement as well, and this is indeed the case in some usages of the word. However, here we follow the HPSG convention of only considering non-subject arguments of the verb to be complements; the subject is treated as a 'specifier', as we discuss in the next paragraph.

[11] In a more complete language analysis, nouns and adjectives can also take complements, but this is too complicated for our brief example here.

[12] In the CFG of Figure 2.2, the VP rule had two optional elements: an NP, to handle transitive and intransitive verbs, and a PP (which is an example of *modification*; we show a way to handle this in HPSG later in this section). The parenthesis notation for optionality allows compactly specifying four different versions of the VP rule, and we are only handling one of these here.

The '1' on the left and right hand sides of the rule is called a 'tag' and indicates that we are constraining the values to have *shared structure* (Pollard and Sag 1994), meaning that we are asserting a constraint that they are identical.[13] It could perhaps have been simpler to instead have [HEAD *verb*] on each side of the rule, but we use this more general version for two related reasons. Firstly, this more general rule can also apply to other parts-of-speech apart from verbs. In particular, we can use it to create NOM nodes from nouns, since they have the same [VAL|SPR −] value[14] as VPs. This means we are taking advantage of linguistic regularities in our data to give a more compact and general grammar.

Additionally, we used this notation to show an example of the parent node taking its HEAD value from its child node, which is an important part of HPSG. As it turns out, when we come up with more grammatical rules to create HPSG versions of phrase structure trees such as those which we saw with CFGs, the parent very frequently takes its HEAD value from one of its child nodes. Encoding this as a core principle of HPSG allows for more concise and elegant accounts of languages. The *head-feature principle* states that the value of the HEAD feature is copied from a privileged child, known as the *head daughter*, to the parent node. This avoids unnecessary redundancy in the grammar, leaving much of the work to be done by a single principle. In the notation here, the head daughter is preceded by a bold **H**. Using this, we could reformulate the rule in (2.29) as shown in (2.30), and the equating of the head values would be handled by the head-feature principle.

$$(2.30)\quad \begin{bmatrix} \textit{phrase} & \\ \text{VAL} & \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix} \rightarrow \mathbf{H} \begin{bmatrix} \textit{word} & \\ \text{VAL} & \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix}$$

We can now use this information to build a small grammar of HPSG which handles a superset of phenomena handled by the CFG grammar in Figure 2.2, although with more generalisability and less overgeneration.

The grammar handles *modification* of noun phrases and verb phrases,[15] when a phrasal category is converted into a similar phrasal category including the modified element. A modifier can precede or follow the phrase which it modifies. Those which precede are known as premodifiers, and include adjectives in English, which modify

[13] Pollard and Sag (1994:19) note that they are often informally described as *unified*, although this is not strictly correct.

[14] Recall that '|' separates feature names, so this means the SPR feature within VAL.

[15] Modifiers (or *adjuncts* in non-HPSG literature) which are involved in modification are frequently contrasted with complements. The distinction is not always straightforward (Somers 1984), but generally, complements are less optional and more intrinsic to the verb semantics, while modifiers can be more freely omitted without changing the core meaning. In the HPSG analyses, complements attach lower in the parse tree.

noun phrases. Postmodifiers, which follow the modified phrase, are often prepositional phrases, which can modify both noun phrases and verb phrases.[16]

The grammar we show uses rules which are fairly standard for HPSG and fall into three categories:

- *Head-complement rules*, which combine a *word* with any compulsory arguments (of which there may be zero) to produce a *phrase*. For us, these rules use the VAL|COMPS feature, and after these rules apply, this is set to a value to indicate that the complement slots are saturated.

- *Head-specifier rules*, which combine a *phrase* instance which has an unfilled specifier slot (encoded on the VAL|SPR feature here) with the appropriate specifier.

- *Head-modifier rules*, which allow for modifiers of noun phrases and verb phrases, either as premodifiers (such as adjectives in English) or post-modifiers (such as prepositional phrases).[17]

In Figure 2.8, we show some feature structure descriptions and succinct corresponding category labels mirroring those of the CFG. These are not needed by the grammar, but provide readable representation of what particular feature structures correspond to, and could be accurately applied to the sample parse tree in Figure 2.5.

However, the grammar in Figure 2.7 also needs a lexicon. As we noted above, a detailed lexicon is a powerful and important tool in a HPSG account of a language. For this small expository grammar, we have a much less detailed lexicon than would be seen in a full-blown grammar, but it nonetheless broadly illustrates some of the important principles – in particular that particular lexical items can select for particular specifiers and complements. Ideally we would also avoid redundancy here by making use of inheritance, but for simplicity here, we fully specify each entry.

A lexical entry here is simply a pairing of a word with its associated feature structure, which will generally include a value for HEAD and a value for VAL.[18] In Figure 2.9, we show some sample lexical entries designed to match those in the CFG

[16]Note, however, that PPs are not always modifiers. In more sophisticated analyses, they can be complements to verbs, nouns or adjectives — for example, the PP in *put the book [on the table]* is often cited as an example of a complement. Somers (1984) (*inter alia*) discusses some tests to distinguish complement PPs and modifiers PPs. For our purposes in this section, we treat prepositional phrases as modifiers.

[17]The reader may notice an asymmetry between the head-modifier rules. Head-modifier rule 1 is fairly standard, but Head-modifier rule 2, for handling adjectival premodifiers has an unusual format where a *word* is a sibling to a *phrase*. This is necessary only because we have not included an adjective phrase category in our very simple grammar.

[18]However, since it is a feature structure it is possible to leave some of these values underspecified — for example, underspecification of the value of NUM could be used to analyse nouns with identical singular and plural forms, such as *sheep*

Head-complement rule 1 (intransitive):

$$\begin{bmatrix} \textit{phrase} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix} \rightarrow \mathbf{H} \begin{bmatrix} \textit{word} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix}$$

Head-complement rule 2 (transitive):

$$\begin{bmatrix} \textit{phrase} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix} \rightarrow \mathbf{H} \begin{bmatrix} \textit{word} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{tr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix} \begin{bmatrix} \textit{phrase} \\ \text{HEAD} \quad \textit{noun} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & + \end{bmatrix} \end{bmatrix}$$

Head-specifier rule 1 (sentences):

$$\begin{bmatrix} \textit{phrase} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & + \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{phrase} \\ \text{HEAD} \quad \begin{bmatrix} \textit{noun} \\ \text{NUM} \quad \boxed{1} \end{bmatrix} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & + \end{bmatrix} \end{bmatrix} \mathbf{H} \begin{bmatrix} \textit{phrase} \\ \text{HEAD} \quad \begin{bmatrix} \textit{verb} \\ \text{NUM} \quad \boxed{1} \end{bmatrix} \\ \text{VAL} \quad \begin{bmatrix} \text{SPR} & - \end{bmatrix} \end{bmatrix}$$

Head-specifier rule 2 (noun phrases):

$$\begin{bmatrix} \textit{phrase} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & + \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{word} \\ \text{HEAD} \quad \begin{bmatrix} \textit{det} \\ \text{NUM} \quad \boxed{1} \end{bmatrix} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & + \end{bmatrix} \end{bmatrix} \mathbf{H} \begin{bmatrix} \textit{phrase} \\ \text{HEAD} \quad \begin{bmatrix} \textit{noun} \\ \text{NUM} \quad \boxed{1} \end{bmatrix} \\ \text{VAL} \quad \begin{bmatrix} \text{SPR} & - \end{bmatrix} \end{bmatrix}$$

Head-modifier rule 1 (postmodifiers):

$$\begin{bmatrix} \textit{phrase} \\ \text{VAL} \quad \boxed{1} \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix} \rightarrow \mathbf{H} \begin{bmatrix} \textit{phrase} \\ \text{VAL} \quad \boxed{1} \end{bmatrix} \begin{bmatrix} \textit{phrase} \\ \text{HEAD} \quad \textit{prep} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \textit{itr} \end{bmatrix} \end{bmatrix}$$

Head-modifier rule 2 (premodifiers):

$$\begin{bmatrix} \textit{phrase} \\ \text{VAL} \quad \boxed{1} \begin{bmatrix} \text{COMPS} & \textit{itr} \\ \text{SPR} & - \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{word} \\ \text{HEAD} \quad \textit{adj} \end{bmatrix} \mathbf{H} \begin{bmatrix} \textit{phrase} \\ \text{HEAD} \quad \textit{noun} \\ \text{VAL} \quad \boxed{1} \end{bmatrix}$$

Figure 2.7: A grammar to account for (2.4)–(2.8)

$$
\text{S} = \begin{bmatrix} \textit{phrase} & \\ \textsc{head} & \textit{verb} \\ \textsc{val} & \begin{bmatrix} \textsc{comps} & \textit{itr} \\ \textsc{spr} & + \end{bmatrix} \end{bmatrix}
\qquad
\text{NP} = \begin{bmatrix} \textit{phrase} & \\ \textsc{head} & \textit{noun} \\ \textsc{val} & \begin{bmatrix} \textsc{comps} & \textit{itr} \\ \textsc{spr} & + \end{bmatrix} \end{bmatrix}
$$

$$
\text{NOM} = \begin{bmatrix} \textit{phrase} & \\ \textsc{head} & \textit{noun} \\ \textsc{val} & \begin{bmatrix} \textsc{comps} & \textit{itr} \\ \textsc{spr} & - \end{bmatrix} \end{bmatrix}
\qquad
\text{VP} = \begin{bmatrix} \textit{phrase} & \\ \textsc{head} & \textit{verb} \\ \textsc{val} & \begin{bmatrix} \textsc{comps} & \textit{itr} \\ \textsc{spr} & - \end{bmatrix} \end{bmatrix}
$$

$$
\text{PP} = \begin{bmatrix} \textit{phrase} & \\ \textsc{head} & \textit{prep} \\ \textsc{val} & \begin{bmatrix} \textsc{comps} & \textit{itr} \\ \textsc{spr} & - \end{bmatrix} \end{bmatrix}
\qquad
\text{V} = \begin{bmatrix} \textit{word} & \\ \textsc{head} & \textit{verb} \end{bmatrix}
$$

$$
\text{N} = \begin{bmatrix} \textit{word} & \\ \textsc{head} & \textit{noun} \end{bmatrix}
\qquad
\text{A} = \begin{bmatrix} \textit{word} & \\ \textsc{head} & \textit{adj} \end{bmatrix}
$$

$$
\text{D} = \begin{bmatrix} \textit{word} & \\ \textsc{head} & \textit{det} \\ \textsc{val} & \begin{bmatrix} \textsc{comps} & \textit{itr} \\ \textsc{spr} & + \end{bmatrix} \end{bmatrix}
\qquad
\text{P} = \begin{bmatrix} \textit{word} & \\ \textsc{head} & \textit{prep} \\ \textsc{val} & \begin{bmatrix} \textsc{comps} & \textit{tr} \\ \textsc{spr} & + \end{bmatrix} \end{bmatrix}
$$

Figure 2.8: Feature structure descriptions mapping to CFG-like node labels for the grammar in 2.7.

grammar in Figure 2.2. We are missing lexical entries for many items, although they look very similar to the existing items.

If we expanded this lexicon fully, we could combine it with the type hierarchy and grammatical rules along with the head-feature principle to give an account of our small subset of English. We can use this to correctly mirror the grammaticality judgments for all of the sentences in Section 2.2.3 which we proposed applying the CFG to, including the agreement and subcategorisation examples.

In Figure 2.10, we show a sample derivation with fully populated feature structures of the same sentence *The pony sleeps* as in Figure 2.5. This derivation uses various head-complement rules and head-specifier rules from Figure 2.7. To create this derivation, we apply the following steps (not necessarily in this exact order, although lower-level nodes must of course be created first):

- Extract the feature structures (FSs) for each word from the corresponding lexical entries
- Apply Head-Complement Rule 1 to the FS of *pony* to give an instance of 'NOM'
- Apply Head-Specifier Rule 2 to the FS of *the* and this NOM node, giving a new NP node
- Apply Head-Complement Rule 1 to the FS of *sleeps*, to give a new VP node
- Apply Head-Specifier Rule 1 to the NP node and the VP node, to give the top-level S node

Whether or not it is clear from this example, HPSG provides one way for creating an account of a language in a scalable and theoretically well-defined manner. HPSG is often considered to be an example of a *unification-based* formalism, since *unification* (finding the most-general feature structure which is consistent with two input feature structures) is used for inheritance and for each rule application. However, Sag *et al.* (2003:56) argue that the term *constraint-based* is preferable, as it is the feature-based constraints specified in the grammar which constitute the theory of a language, while *unification* is simply a procedure for solving identity constraints.

More Advanced HPSG

The account we have presented here is only the absolute basics of HPSG, with the minimum presented to make this thesis comprehensible. There is far more to the formalism than we have presented here. For example, the rules can be much more powerful and complex than those in Figure 2.7. The handling of SPR and COMPS is particularly important in HPSG. Here, we presented the approach from Sag *et al.* (2003: Chapter 3), which is too inflexible and simplistic to be a general solution, and

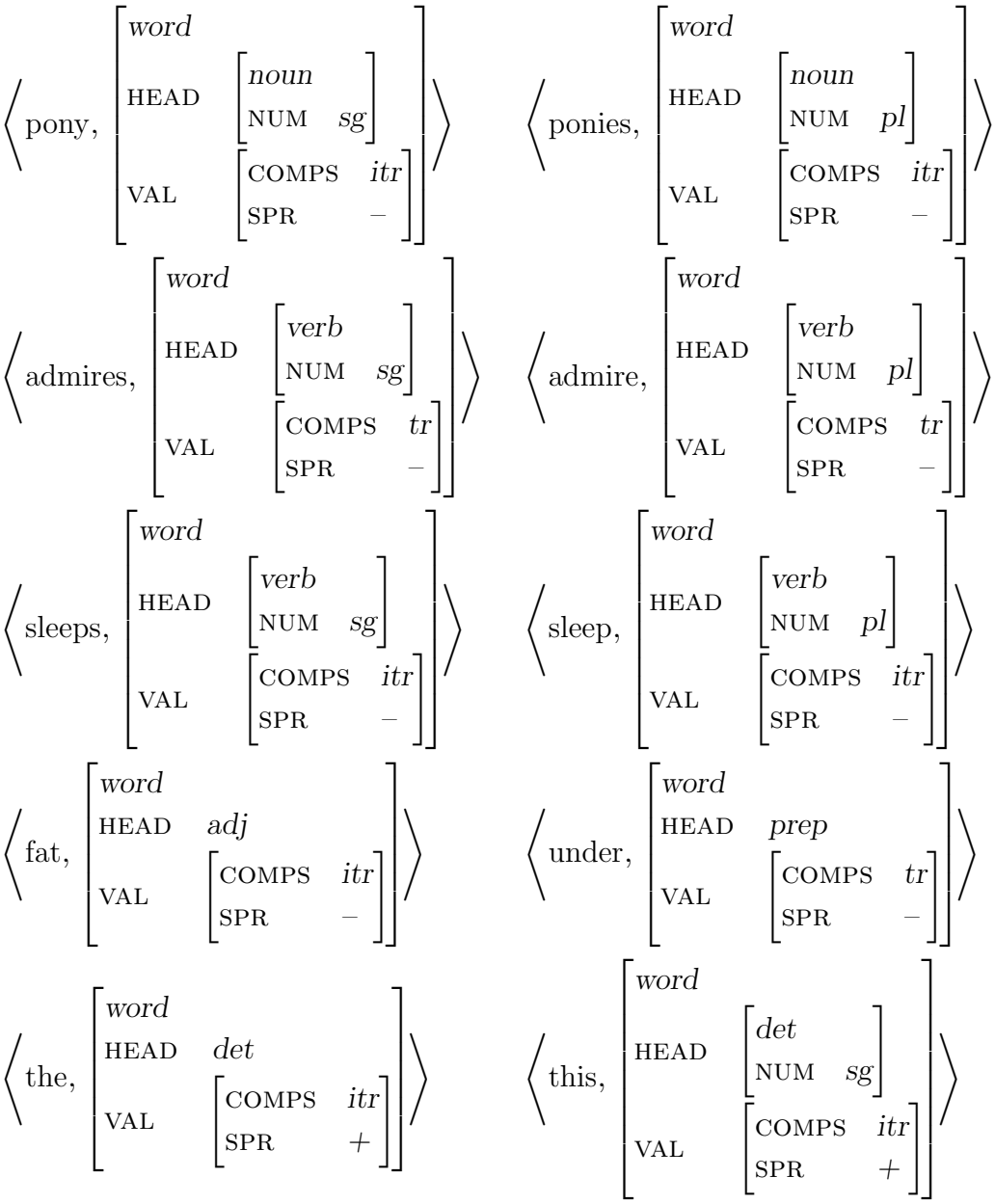


Figure 2.9: Sample fully-specified lexical entries mapping for the grammar in 2.7.

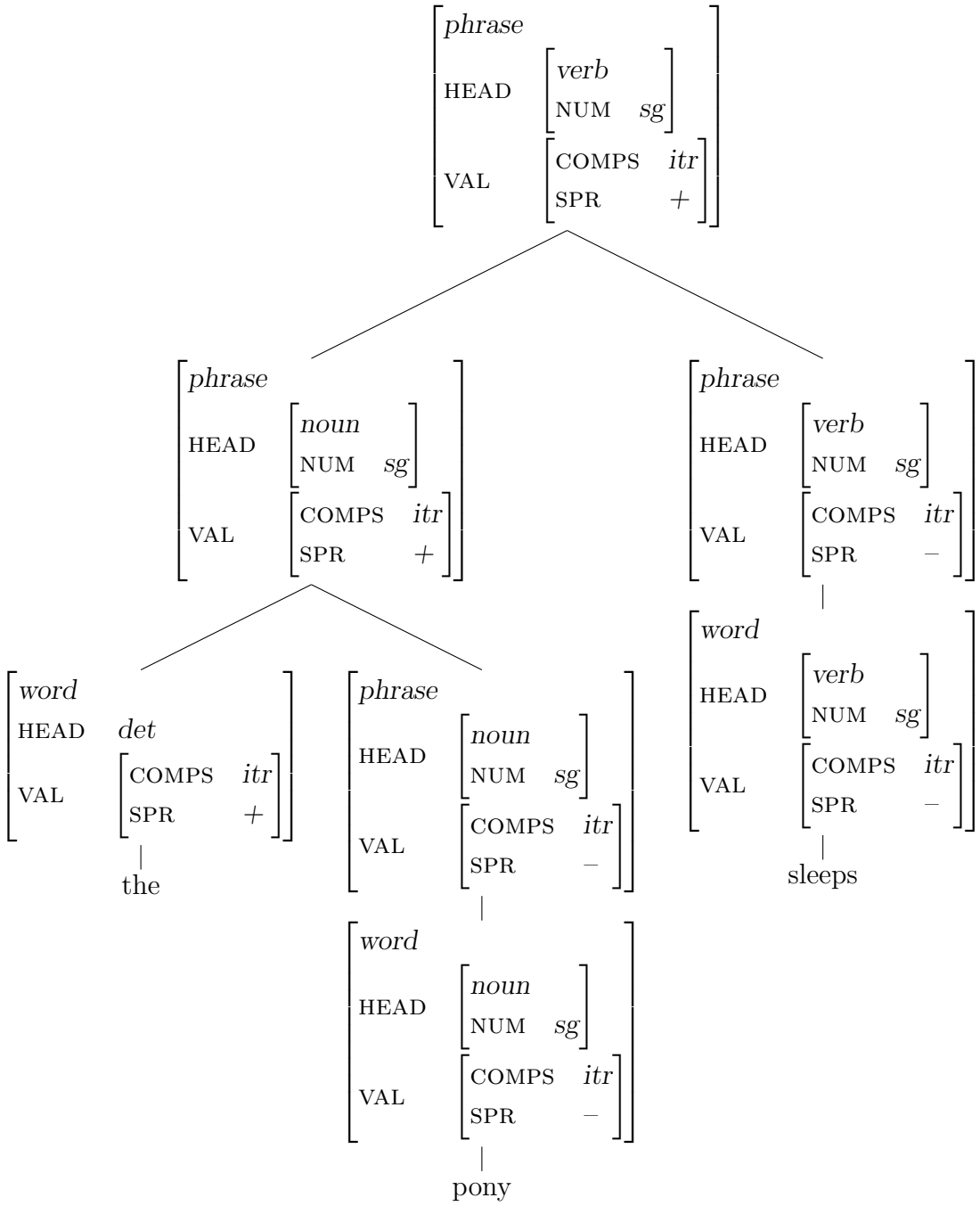


Figure 2.10: Analysis for *the pony sleeps.*

is superseded in subsequent chapters by a more powerful approach which is more likely to be used in real grammars.

With the addition of some extra features, it is possible to write rules to handle syntactically complex phenomena such as passivisation (e.g. *The pony was admired*) and *wh*-questions (e.g. *Who admired the pony?*). In contrast to theories such as Government and Binding, these phenomena can be handled without rules postulating movement in the surface realisation, since the feature structures are powerful enough to encode this information (Pollard and Sag 1994). Additionally we have not covered some standard features which are conventionally used on *expression*-like feature structures in HPSG, including PHON(OLOGY), DTRS (daughters) and SYNSEM (syntax/semantics) (Pollard and Sag 1994). Indeed, we have not presented an account of the semantics at all, but we return to this question in Section 3.5.4.

The presentation of the lexicon in Figure 2.9 was somewhat misleading — a more detailed and realistic exposition of building a lexicon is presented in Sag *et al.* (2003: Chapter 8). In a real grammar, there would generally not be such a large amount of repetition between lexical entries. Instead, there would be a type hierarchy for lexemes with different types, at a finer level of granularity than simply noun and verb lexemes, to account for the different syntactic and semantic behaviour possible. For example, we would have different lexeme subtypes for the possible verb subcategorisations. The lexical entries would then use inheritance from these types to minimise repetition of information. If we fully expanded each feature structure, we would get a similar result to what we showed in Figure 2.9 — although of course the feature structures would be much more complex in a real grammar. The lexicon would also not generally have separate entries for the various inflected forms of nouns and verbs encoding the singular or plural agreement. Such *morphology* would generally be handled by *lexical rules* (Pollard and Sag 1994; Briscoe and Copestake 1999) which create the appropriate inflected forms.

HPSG, including the advanced aspects we have not covered here, is the basis for a range of large-scale computational grammars of various natural languages, most notably English (Flickinger 2000) and Japanese (Siegel and Bender 2002). The former is a grammar known as the English Resource Grammar, which is integral to this thesis, and which we explore in some detail in Section 3.1.1.

It is worth making it explicit at this point that there are many other syntactic formalisms (some of which we have mentioned in passing) which seek to address the deficiencies with simple accounts of language based on CFGs, in order to provide a more complete account of natural language syntax. HPSG is by no means the only formalism to propose more elegant handling of language and solutions to the problems with CFGs we noted in Section 2.2.3. Some other theories are Lexical-Functional Grammar (LFG; Bresnan 2000), Combinatory Categorial Grammar (CCG; Steedman 2000), Lexicalized Tree-Adjoining Grammar (LTAG; Joshi and Schabes 1997) and Government and Binding Theory (GB: (Chomsky 1982)). All of these approaches have strengths and weaknesses but generally have mechanisms to explain a wide

range of natural language phenomena, so provide a good account of natural language syntax. In any case, there is a substantial overlap of linguistic theory informing different advanced syntactic theories – Sells (1985) notes that LFG, GB and Generalised Phrase Structure Grammar (Gazdar 1985), which is a predecessor of HPSG, all have a common ancestry and a shared body of central concepts. We have only discussed HPSG in detail due its central role in this thesis.

2.3 Machine Learning

Machine learning[19] is a broad sub-field of computer science which involves computationally discerning meaningful patterns from data, and applying those patterns to new data (Hastie *et al.* 2001; Witten and Frank 2005). In this thesis, we are primarily concerned with *supervised learning* — specifically the variety described in Witten and Frank (2005) as "classification learning", since it involves applying classifications.

In this paradigm, we have a *training set*, composed of instances which have somehow been assigned a *gold-standard* correct classification or label. This gold-standard label is the reason we call this "supervised", as we are explicitly providing the learner with an indication of the correct classification. In the training phase, we use some learning algorithm to discover how to automatically apply similar classifications to new data, in the process creating a classifier. The generalisations the algorithm learns from the training data are referred to as the *model*, since it is designed to model patterns in the underlying data.

When evaluating the performance over new data, we would like an indication of how well our algorithm has generalised from the training set as it built the model. To achieve this, we apply the trained algorithm to a *test set* and measure how accurately it can produce the labels on these instances. However, if we are experimenting with a number of different ways of training learning algorithms and we repeatedly evaluate these changes on the test set, there is a risk of *overfitting*, or too closely mirroring idiosyncrasies of this test set. For this reason, it is fairly common practice to use a *development set* in the experimentation phase and preserve the test set for a final evaluation after one set of good training parameters has been found.

The data instances from the training and test sets encode particular salient properties as *features* (or "attributes" in the terminology of Witten and Frank (2005)). The process of determining the most useful features to extract from the raw data and attach to each instance is known as *feature engineering* and is an important part of machine learning.

[19] Alternate terms with similar meanings include *statistical learning* and *data mining*

2.3.1 Machine Learning within NLP

Most modern mainstream NLP research makes use of quantitative methods of some sort; a fairly comprehensive survey is presented in Manning and Schütze (2000). Quantitative methods include a broad range of techniques of statistically analysing language, including machine-learning techniques. It has been argued (Abney 1996) that these statistical methods are not merely a convenience employed by NLP researchers, but reflect an underlying statistical nature of language itself.

We present here only those machine learning methods which fall within the supervised learning paradigm, and which are important to this thesis. Specifically, we are primarily interested in *maximum entropy modelling* (Berger *et al.* 1996; Ratnaparkhi 1998). One of the reasons for its attractiveness to NLP researchers is its ability to handle large feature set sizes in the range of several hundred thousand (Malouf 2002). *Maximum Entropy* ("MaxEnt") techniques rely on the principle that the best model for the data is considered to be the one which makes a minimal amount of assumptions beyond what is licensed by the input training data, so is the most "uniform" — that is, the one with the maximum entropy.

Here, "entropy" is used in its information-theoretic sense, and refers to the amount of information inherent in some random variable. A higher entropy means we have to supply more information, on average, to convey the value of that random variable — in other words, it is more uncertain. So, the best model of the training data is the one which is most uncertain, but which is still consistent with the training data. Maximising entropy formalises the notion of creating the most uniform model (Berger *et al.* 1996).

A MaxEnt model consists of a set of *feature functions* and the associated weights assigned to each of those features. A feature function takes as input the data associated with the instance as well as the class label assigned to that instance, and maps it to some numeric value.[20] This is often boolean or integer-valued, although it is also possible to work with real-valued feature functions (Malouf 2002).

Assume we have a particular instance we are trying to classify, and K feature functions. From the mathematical definition of maximising entropy, it is possible to derive the probability p of a particular class label under this MaxEnt model given some corresponding vector of data (Berger *et al.* 1996):

(2.31)
$$p_\lambda(c|x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_{i=1}^{K} \lambda_i f_i(x, c)\right)$$

[20] The term "feature function" is often abbreviated as "feature" in the maximum entropy literature, but differs slightly from what is used in the broader machine learning literature (and the previous section), since it depends on the target class label as well as the information extracted. We use "feature function" when it is intended to reflect the more specific meaning from ME literature.

where c is the class, x is the input data, $f_i(x, c)$ denotes the ith feature function and $Z_\lambda(x)$ is a normalisation factor which is constant for a given context x. We do not need to calculate Z_λ if we are only interested in the relative values of p_λ for different values of c (for a full explanation, see Berger *et al.* (1996)), which is the case when applying the model to a given instance, as the context remains constant. This is simply a formulation of a log-linear model, which is so-named because the log of the probability is linear with respect to the feature weights multiplied by the values. Maximum entropy estimation is one kind of log-linear modelling. It is also a kind of *discriminative model*, or a *conditional model*, since it models the conditional probability $p(c|x)$. These are contrasted in the literature with *generative models*, which model $p(c, x)$. See Ng and Jordan (2002) for a discussion and empirical evaluation of some of the differences.

It is still necessary to estimate the value of the λ_i parameters. A variety of methods have been used to estimate these parameters. Malouf (2002) presents a comparison of these methods. Generally, they make use of the fact that maximising the entropy of the distribution is equivalent to maximising the conditional log-likelihood L of the observed distribution $\tilde{p}$, which is given by Berger *et al.* (1996):

(2.32)
$$L_{\tilde{p}}(p_\lambda) = \sum_{c,x} \tilde{p}(x, c) \log p_\lambda(x|c)$$

However, it is not possible to analytically maximise this with respect to the λ parameters. Instead, the standard methods start off with some parameter values and iteratively update these based on the previous set of values in a way which leads towards a maximum, until the change in log-likelihood is less than some threshold. One of the earlier methods to achieve this was Improved Iterative Scaling (Berger *et al.* 1996), but more recently *gradient-ascent* methods such as *conjugate gradient* and *steepest ascent* have gained wider use. These methods attempt to find the maximum by calculating the direction in which the gradient of L is steepest. The Limited Memory Variable Metric (LMVM) method (Benson and More 2001) also takes into account the second derivative of L for faster convergence, and carefully manages the potentially prohibitive memory requirements of such an approach. These are amongst those investigated by Malouf (2002), who finds that LMVM and the other gradient-based methods outperform iterative scaling methods in terms of training time and sometimes accuracy of the learned model. Many further optimisations to the process have been proposed and implemented, such as adding Gaussian priors (Chen and Rosenfeld 1999). This is used to *smooth* the model towards the uniform model in order to avoid overfitting, and means that the objective function in (2.32) is modified to have an extra term dependent on the pre-specified variances.

2.4 Parsing Natural Language

Parsing (Manning and Schütze 2000, Chapter 12; Jurafsky and Martin 2000, Chapter 12) is the task of deriving syntactic analyses from plain text. It involves assigning a syntactic analysis which conforms to some grammar, such as those in Figures 2.1 and 2.10, to a sequence of characters.

Generally, this was a straightforward process for the expository syntax examples in Section 2.2. However, as we might expect for natural language, this is not always as simple as it might seem from these basic examples. Recall from Section 2.2.3 that even the very simple grammar we showed was ambiguous for prepositional phrase attachment, so even in those highly-constrained circumstances for a fairly short sentence, we have ambiguity.

Additionally, to parse anything more complicated than carefully constructed toy sentences, we need a comprehensive grammar of some kind which has good *coverage* — that is, it can create parse trees for a large percentage of naturally occurring sentences. A broad-coverage grammar needs to include a wide range of syntactic constructions, as well as including many lexical items, which frequently need to be ambiguous to cover rarer usages. For example in the sentence *The pony sleeps* we analysed in Figure 2.10, we analysed *pony* as a noun, when it could also be a verb (*He needs to* ***pony*** *up $500*), and *sleeps* which could be a count noun (*Only three* ***sleeps*** *to go*). A frequently-quoted example of these lexical ambiguities translating to genuine parsing ambiguities is the five-word sentence

(2.33) *Time flies like an arrow.*

There is an obvious interpretation for a native speaker reading the sentence, with *flies* as the main verb of the sentence and *like* as a preposition heading the prepositional phrase which modifies the verb. However *flies* can also be a noun, while *like* can also be a verb. In English, it is possible to join together sequences of nouns in *noun compounds*, and a grammar of even moderate coverage would need to be able to apply this rule productively.[21] So, *Time flies* could also be analysed as a noun compound, becoming the subject of the verb interpretation of *like*, giving the same parse tree shape as in the obvious reading of

(2.34) *Fruit flies like a banana.*

(although this also permits the alternate "flying fruit" reading).

It is easy to see that increasing the number of grammar rules and increasing the lexical ambiguity would each on their own substantially increase the number of possible parse trees for a sentence. When they are applied to longer sentences observed in real-world language, most sentences become highly ambiguous.

[21] Meaning it can apply in a broad range of circumstances

This ambiguity makes parsing difficult for two reasons. Firstly, there is the problem of correctly selecting the correct parse from the range of parses allowed by the grammar. Secondly, enumerating the large number of parses and selecting correctly from them can make it challenging to store all parse tree information in memory and parse sentences in a tractable time.

2.4.1 Parsing using Treebank-Derived Grammars

Being able to parse sentences according to a grammar of a language also presupposes the existence of such a grammar. One intuitively obvious approach is to manually create a grammar of a language, which, as we explain in Section 2.4.3, is indeed the strategy of some researchers. However, many approaches induce grammars (for some definition of the term) automatically from annotated data, which we explore in this section.

Treebanks

While it is possible to induce grammars from unannotated data (Klein and Manning 2002; Clark 2001), most work on parsing relies on some pre-existing *treebank*, which is a corpus of sentences annotated by humans with phrase structure trees.

Many treebanks now exist, and they are introduced at relevant points throughout this thesis. However, a particularly early, important and influential treebank is the Penn Treebank (PTB: Marcus *et al.* 1993). This corpus contains 4.8 millions tokens of text which have been manually POS-tagged; of these, 2.9 million have been manually marked up with phrase structure trees (PSTs). The raw text for this comes from a range of sources. However, most of the parsed text (2.2 million words) comes from just two of these: the complete raw text of the Brown corpus (Francis 1979), a mixed genre corpus including fiction, biography, newswire and government documents totalling 1.1 million tokens, and a collection of articles from the *Wall Street Journal* (WSJ). It is the section containing the WSJ articles which is used for most published work on the corpus.

The annotation process for the PTB both for POS-tagging and for syntactic annotation, involved an automatic initial phase, followed by post-correction by humans (Marcus *et al.* 1993). The POS-tags and PSTs look very roughly similar to CFG-based analyses such as those we showed in Section 2.2.3. However, apart from linguistic utility, one of the concerns for Marcus *et al.* was to have the text efficiently annotated, as well as maximising inter-annotator agreement, so they enacted a number of compromises to simplify the analysis, assisting with both of these goals.

One noticeable simplification is a comparatively small tagset compared to earlier work, with 36 items plus punctuation tags, compared to 87 atomic tags for the Brown corpus. The syntactic bracketing is also comparatively shallow and simple. Noun phrases are generally flat, with no equivalent of the 'NOM' category (for nominals

without determiners) which we used in Section 2.2.3, and possibly multiple modifier PPs attached at the same level. The flatness of NPs also extends to noun compounds, so that even multi-term compounds such as *baseball bat rack* have only one level of structure.[22] For VPs, there is another linguistically relevant distinction which is not made. Annotators are permitted to remain agnostic about the complement/modifier distinction when it is not clear, meaning that very frequently it is not clear whether a PP is a modifier or complement to some verb (Marcus *et al.* 1993).

While the PTB is not intended to deliberately encode one particular formal theory of grammar, these decisions in its design mean that it implicitly endorses a lightweight theory of syntax postulating fairly flat phrase structures. The treebank does explicitly represent some long-range dependencies in the form of 'traces' showing a theoretical underlying location of a relocated token but these are often ignored by treebank users.

Probabilistic Treebank Parsing

Many different parsers using different techniques have been applied to the WSJ section of the PTB, and it is not our intent to exhaustively cover them here, but rather to give an idea of the general techniques involved. The types of parsers we discuss all start with relatively little linguistic knowledge, and instead gather statistics from the treebank which can be used to create new parse trees. In this sense, these parsers in the learning phase are inducing grammars, although only in some cases do the grammars bear much resemblance to those we discussed in Section 2.2.3. In most cases here, these parsers learn from and are tested on at least the WSJ section of the PTB, and thus fall under the broad banner of treebank parsing. All of these solve the ambiguity problem by using a statistical model to prefer one parse tree over another, returning the highest-scored tree. Except where noted, they are *lexicalised* — that is, they use statistics based on the lexical items themselves in addition to statistics based on POS-tags (rather than POS-tags exclusively), and have some notion of headedness, so that the key item from some phrase can be used in statistics of parent nodes in the parse tree.

One of the earlier works in this space is the parser of Magerman (1995), which learns a decision tree from the treebank. This implicitly encodes grammar rules and can be used to calculate the probability of different analyses, so it can be used to search through the large space of possible parses for a given sentence, and return the highest probability tree. An alternative approach is that of the Collins (1996) parser, which maps between trees and dependency-like structures denoting syntactic paths between the NPs. It learns a probability distribution for the possible dependency triples, and uses these probabilities to determine the licensed parse trees and subtrees at any point, and to rank them in order to produce the best tree.

The unlexicalised Charniak (1996) parser, in contrast to the previous methods, explicitly induces a grammar which is somewhat recognisable as something we have

[22] However, see Vadas and Curran (2007) for an approach which adds internal structure to NPs

seen before. From the treebank, it learns a *probabilistic CFG* (PCFG). This is in principle much like the CFGs we saw in Section 2.2.3 (albeit much larger), but with each grammatical production augmented with a probability, enabling the calculation of a probability for each tree as the parse tree is built (making this a kind of generative model). The statistics are gathered by recording the productions extracted from each tree in the training treebank. In the training phase, each production is added to the grammar, and assigned a probability based on the number of times the rule was applied, divided by the number of rule applications in total which expanded the same non-terminal. After training on the PTB WSJ training corpus, the grammar had around 10,000 rules, including 6000 singletons. The parser then uses a standard HMM Viterbi algorithm with PCFG extensions (Kupiec 1992; Manning and Schütze 2000, Section 11.3.3) to apply the induced PCFG. The Charniak (1997) parser is an extension of this to add lexicalised statistics. It uses the bare PCFG as a pre-filter in the parsing stage but also gathers and applies statistics based on tokens, phrasal heads and parent nodes, giving parsing accuracy slightly exceeding the aforementioned Collins and Magerman parsers.

Also noteworthy are the Collins (1997) family of parsers, which similarly induce a lexicalised PCFG. The first parsing model is closely based on Collins (1996) but uses a generative PCFG, while the other models described add statistics to cover information about certain long-range dependencies and verb subcategorisation, and simultaneously improve parsing accuracy. The Charniak (2000) parser is based on similar principles to those of Collins (1997) and Charniak (1997), but uses a more sophisticated probability model in the PCFG. The probability for a given parse (which is used to determine the best parse) is calculated based on particular elements external to each constituent. Charniak uses techniques inspired by the maximum entropy techniques discussed in Section 2.3.1, and is able to obtain better probability estimates by having more information on which to base these estimates, and better *smoothing* to handle data sparseness.

In common between all of these grammar induction processes mentioned so far is that after creating the grammar induction algorithm, the grammar and associated probabilities can be extracted automatically from the treebank. No expensive human labour is required to generate the grammar itself. On the other hand, for these methods to be possible at all, it is necessary for a large-scale treebank to exist beforehand, and this treebank itself represents a very large amount of embodied labour on the part of the treebank annotators. Another characteristic in common between these induced grammars is that they are shallow, in that they generally do not resolve long-distance dependencies, largely reflecting the structures in the source treebank and ignoring the aforementioned trace information (although later work (Collins 1999; Johnson 2002) does introduce some support for this), and do not map parse trees to an extended semantic representation (Cahill *et al.* 2008).

2.4.2 Inducing Deep Grammars from Treebanks

The work outlined in Section 2.4.1 depends on directly learning how to reapply the syntactic structures present in the source treebank. In the case of the PTB, this is a fairly formalism-neutral syntactic representation. However, there has also been some research on taking advantage of the PTB to apply syntactic formalisms with more clearly defined theoretical bases, which have the ability to create deeper linguistic analyses and output semantic representations.

Xia (1999) presents work on systematically transforming the structures from the PTB into trees compatible with LTAG. An important part of this is a *binarisation*[23] step applied to the source trees, to ensure that certain nodes in the trees have at most two children. This involves selecting the *head child* using a *head percolation table*, which lists allowable heads for syntactic node labels — for example, a syntactic node labelled 'VP' can have 'VP', 'VB', 'VBP', 'VBZ', 'VBD' or 'VBN' as its head child. Once the head child is determined, it is possible to systematically insert additional nodes in the subtree to create a binary-branching structure. For example, every modifier after the first is attached to a newly inserted node with the same category as the root of the subtree, so that a VP subtree with a verb and two modifier PPs as children such as '[VP [VBZ PP_1 PP_2]]' would be transformed into '[VP [VP [VBZ PP_1] PP_2]]'. As part of the binarisation process, the algorithm also adds distinctions between modifiers and arguments (although it is not clear how accurately this can be done automatically) and transforms conjunctive phrases. From these transformed trees, it is now possible to extract *elementary trees* which are fundamental objects used by the LTAG formalism and can be used as an LTAG treebank after filtering out linguistically invalid items.

The work of Hockenmaier and Steedman (2002) is roughly parallel, but targets CCG, outlining some systematic transformations which are applied to the Penn Treebank to create a CCG treebank. The processes include a similar binarisation step as Xia (1999), inserting an extra level of structure into noun-phrases and systematically remapping of node labels to CCG categories, along with some data cleaning.

These techniques have also been applied to constraint-based formalisms. Cahill *et al.* (2002) present an LFG-based approach where *f-structures*[24] are derived from the PTB trees. This is achieved by heuristically annotating all PTB nodes with "proto-f-structures" (which are less fully-specified variants) and sending these annotations to a constraint solver, producing an f-structure for the sentence. As part of this, like in the other work mentioned here, the phrasal heads are identified, although in this

[23] This term is not explicitly used by Xia but is used in the work listed in the following two paragraphs.

[24] In LFG terminology (Bresnan 2000), *c-structure* is the "constituent structure" or "categorial structure" corresponding to the surface syntactic form of the sentence, and *f-structure* is "functional structure" which seeks to be more language-neutral, representing the elements of the sentence and how they relate to each other irrespective of their ordering within the sentence. These f-structure features are more semantic in nature such SUBJ(ECT), PRED(ICATOR) or TENSE.

case there is no binarisation phrase. The f-structures are used in both a pipeline architecture and an integrated architecture. The latter is closer in spirit to the other grammar induction work outlined here. A PCFG is induced from the annotated trees, with each node label set to the combination of the original CFG node label and the string corresponding to the proto-f-structure. This annotated PCFG can be applied to new sentences, and the resulting structures are sent to a constraint solver to create a well-formed f-structure for the sentence. They do not claim that this annotated PCFG is strictly an LFG implementation, although it has many characteristics in common with one, since the annotations are based on LFG principles.

A grammar induction analog in the framework of HPSG, another constraint-based formalism, is the Enju grammar of Miyao *et al.* (2004). As in most of the previously-described work, there is a phase of binarisation and encoding of argument/modifier distinctions. The trees from the PTB are automatically annotated with sufficient information to turn them into partially specified derivation trees using some heuristics. This includes specifying HPSG rule schemata (such as Subject-Head, Head-Complement, Head-Modifier) and HPSG categories for each corresponding PTB node, and annotating with some salient feature values relating to verb subcategorisation and relativisation features. Subsequent work (Miyao and Tsujii 2005) uses the induced HPSG-based grammar to parse sentences. For parse-ranking, this work uses a maximum entropy model trained by creating a parse forest of many licensed trees from the induced grammar corresponding to the sentences in the treebank. For each sentence, the tree from this forest which matches the HPSG-augmented treebank tree is the gold tree, while the remaining trees in the forest provide sources of negative evidence which are used in the model training for optimising the objective function to calculate the MaxEnt feature weights in training.

2.4.3 Deep Parsing using Handcrafted Grammars

For the simple induced grammars described in Section 2.4.1, the linguistic knowledge acquired by the parser in order to apply syntactic analyses comes from the knowledge embodied in the treebank on which the grammar was trained. This knowledge in turn comes from the linguistic judgements of the skilled treebank annotators. So, treebank parsers are learning to reproduce these annotators' judgments to some extent.

In the approaches outlined in Section 2.4.2 for inducing deep grammars, most of the linguistic information again comes from the treebank, however this is often augmented with other additional linguistic information (for example, in the form of parse tree annotations) representing a further layer of linguistic expertise which is then reproduced by the parser. In other words, the sources of information available in these hybrid approaches are the linguistic judgements of the treebank annotators as well as the expertise of the researchers who add formalism-specific information.

Thus, we could interpret these hybrid approaches as a form of lightweight *grammar engineering*, or manual creation of a grammar of a language.

There is much more scope for grammar engineering in the creation of a computational grammar of a natural language. At the opposite end of the spectrum to the completely treebank-derived grammars of Section 2.4.1, we have the approach of creating completely hand-crafted grammars. Rather than deriving the grammar rules from a treebank, they are specified manually by grammar engineers. Often the lexicon is also manually specified. The source of linguistic knowledge in these grammars, then, is primarily the grammar engineer's expertise. The main exceptions to this are in the subtask of parse ranking, which we discuss later in this section, and in robustness techniques such as the handling of words outside the lexicon of the grammar, which we discuss in Section 3.3.6.

In contrast to creating a large-scale treebank, explicitly creating a grammar of a language allows a single point-of-control for the handling of syntactic phenomena. If a rarer phenomenon is encountered in some test corpus, such as the type of construction in *the bigger the better*, the grammar engineer can decide on a sensible analysis for it[25] or even choose to not handle it at all. On the other hand, in creating a treebank, the decision about handling analyses is enforced by ad hoc annotation guidelines and this decision is fossilised in the treebank, and it is not possible to choose to ignore particular difficult or unusual phenomena.

Creating a grammar of a language which has reasonable coverage and can create meaningful parse trees and semantics is a very labour-intensive task. Nonetheless, a range of efforts, some of which are related, have created broad-coverage grammars for several languages.

Often these grammars are designed explicitly to represent a particular syntactic formalism — for LFG, grammars for many languages have been developed as part of the ParGram project (Butt *et al.* 2002). When that original paper was published, they had created grammars for English, German, French, Japanese, Norwegian and Urdu, although the final three were all relatively small. The English grammar achieves coverage of around 75% over *WSJ* newswire text, or 100% if some robustness techniques (accepting less accurate or fragment analyses) are used in addition (Riezler *et al.* 2002). Over newswire text, technical manuals and spoken dialog transcriptions, the Japanese grammar achieves coverage of 87–92% using the grammar proper and 97–99% using the same robustness techniques (Masuichi *et al.* 2003). It is not entirely clear what coverage the other grammars can achieve on similar text, but the Norwegian and Urdu grammars are described as relatively immature, so may not have achieved broad coverage at that time.

[25] They can even change their mind later in some cases. In Section 3.3.4, we discuss the creation of *dynamic treebanks* for a particular family of handcrafted grammars. These treebanks must closely match the grammar, but this treebanking process allows relatively painless updates to the treebank if some rule is changed in the corresponding grammar.

There are parallels to this work in the HPSG space. The DELPH-IN consortium[26] is associated with several parsers and grammar development tools, which we cover in more detail in Chapter 3. The most-extensively developed grammars, which have the broadest coverage, are those of English, Japanese and German (these were briefly mentioned in Section 2.2.4). The grammar of English is the English Resource Grammar (ERG; Flickinger 2000). As the focus of this thesis, we explore it in some detail in Section 3.1.1. For now, we will note that it achieves relatively high coverage over various domains — for example, 85% over Wikipedia (Flickinger *et al.* 2010). GG (Müller and Kasper 2000), the grammar of German, achieves 74–85% coverage over various test corpora (Crysmann 2005), while JaCy, the Japanese grammar has 78–94% coverage over two test domains (Siegel and Bender 2002).

Some points are worth making about the coverage figures. One is that the grammar developers seek to make the grammar as constrained as possible, while still permitting the grammatical sentences from the test suite, for two reasons – to avoid permitting ungrammatical sentences, but also to avoid ambiguity, particularly where this is spurious (such as different attachment points for pre-modifiers and post-modifiers corresponding to no meaningful semantic difference). We have already noted that a very simple grammar covering all sentences is trivial to create but also completely useless. There is thus a tension between avoiding overgeneration and maximising coverage, and these various grammars may have been optimised differently and made different tradeoffs in this regard. Secondly, the fact that a grammar is able to parse a sentence does not necessarily mean that it is able to *correctly* parse it — this evaluation requires a human, although in Chapter 4, we will see some concrete figures for this for the ERG.

We discussed that in treebank parsing, the parser simultaneously learns a grammar and the required statistics, so that it can probabilistically apply that grammar to new sentences and use the statistics to determine the best parse. We have not yet addressed the equivalent question for parsing with handcrafted grammars. That is, both treebank grammars and hand-crafted grammars are ambiguous for most sentences of significant length. In the case of treebank grammars, the means of resolving ambiguity comes from the treebank itself. However, handcrafted grammars are not necessarily as intimately tied to treebanks. We have not yet explained how it is possible to actually select the best parse from those licensed by the handcrafted grammar, which is necessary when ambiguity is present and we wish to make use the outputs of the parser.

There are a number of different approaches to this parse selection problem, but they generally make use of some annotated data. In some cases (Johnson *et al.* 1999; Toutanova *et al.* 2005; Zhang *et al.* 2007), this is a treebank which is directly compatible with the grammatical formalism. These treebanks can be different in nature to the PTB — in particular, the Redwoods Treebank of HPSG (Oepen *et al.*

[26] http://delph-in.net

2004) used by Toutanova *et al.* and Zhang *et al.* is radically different in its construction method, as we explore in more detail in Section 3.3.4. In other cases, syntactic analyses are derived partially from an incompatible treebank such as the PTB. This is the approach of Riezler *et al.* (2002) and Hockenmaier and Steedman (2002), which we explore in more depth in Section 2.7. In either case, from these treebanks, it is possible to build PCFG-like generative models, as investigated by Toutanova *et al.*, or discriminative log-linear models (these are in the same general class as the MaxEnt models of Section 2.3), which are evaluated by Johnson *et al.*, Riezler *et al.*, Toutanova *et al.* and Zhang *et al.* These can be used to select the preferred parse tree in the parsing stage as the one with the highest-probability according to the model. The details of training and applying parse selection models for HPSG-based DELPH-IN grammars is of some importance to this thesis, and we explore this in more detail in Section 3.3.5.

2.5 Supporting Tools for Parsing

There are many other interesting subproblems in NLP. Some are objects of research in their own right, but are also important as supporting tools for parsing tasks. It is this latter usage which we are concerned with here.

2.5.1 Part-of-speech Tagging

POS-tagging is the automatic assignment of (sometimes multiple) reasonably coarse-grained POS-tags to each token in a sentence. POS-tags are roughly intended to group together items with similar syntactic function, but different tagsets have had different design criteria — for example, the Penn Treebank tagset is deliberately coarse-grained, so it attempts to avoid lexically recoverable distinctions, such as those between the copular verb *be* and other verbs (Marcus *et al.* 1993). This gives a tagset of 48 tags including punctuation. Of these, six are for verbs (reflecting distinctions of person, number and tense) and four are for nouns (depending on number and whether it is a proper noun). There are of course many other ways of dividing up words into sets of POS-tags for English (and countless more for other languages) depending on the design criteria, intended purpose, and what is considered as syntactic similarity. We noted in Section 2.4.1 that the Brown corpus has 87 atomic tags; these can also be compounded, for e.g. clitics such as *'ll* in *He'll*, giving 186 tags. A comparison of these and other tagsets is presented in MacKinlay (2005). The influence of the Penn Treebank has been such that the PTB tagset has become a de facto standard for English, although work making use of other tagsets as parsing components does exist (e.g. Briscoe *et al.* 2006).

The various tagging algorithms which are available (Ratnaparkhi 1996; Toutanova *et al.* 2003; Giménez and Màrquez 2004; Brants 2000 *inter alia*) mostly operate in

similar ways, using machine learning algorithms applied to features derived from local context of the word being tagged. A large amount of the disambiguating information comes from the text of the word itself — if we see *fluctuation* in running text, we can be fairly confident that it is a singular common noun regardless of context. However, many words, particularly common words such as *saw*, are ambiguous (as we discussed in Section 2.2.3), and there are also unavoidably unknown words, which were not observed in the training data. In both of these cases, the local context provides powerful disambiguation information. This usually comes from the tags and/or tokens in a window of two tokens on either side (e.g. it is likely that a token following *the* is a noun), as well as from prefixes or suffixes of the token itself (e.g. a suffix *-tion* is also strong evidence that we have a noun).

2.5.2 Supertagging

Supertagging (Bangalore and Joshi 1999) is closely related to POS-tagging (or arguably a special case of it) in terms of the techniques used, but attempts to encode finer-grained lexical distinctions by assigning *supertags*, instead of the broader POS-tags. Unsurprisingly, the set of supertags tends to be larger — for example Curran and Clark (2003) use 398 tags, while many PoS-taggers mentioned in the previous section are optimised for the 48-tag PTB tagset. However in terms of raw tagset size, there is not always a sharp distinction between the two. The CLAWS7 tagset (Garside *et al.* 1997), at 146 tags, is roughly 3 times larger than the PTB tagset, and more than one third of the size of the tagset of 398 tags used by Curran and Clark. There is also a corresponding increase in per-token ambiguity of the supertags (Bangalore and Joshi 1999). Perhaps a more important difference than tagset size is that the tagset in supertagging is more closely tied to a particular grammar and explicitly designed for parsing using that grammar. Thus, supertags generally have a one-to-one correspondence with some fine-grained lexical classes in the grammar — such as CCG lexical categories (Curran and Clark 2003) or lexical types in the ERG or JaCy (Dridan and Baldwin 2010). POS-tagging and supertagging are nonetheless sufficiently closely related that very similar techniques work in either case, although accuracy figures tend to be lower than with PTB-based taggers due to the increased difficulty of assigning the finer-grained tags.

2.6 Domain Adaptation for Parsing

The *domain* of some piece of text refers to its subject matter as well as its style of language, dialect and level of formality. It is surprisingly difficult to give a succinct or definite delineation of exactly what constitutes a domain, and where the boundaries of

given domain lie. Nonetheless, given a pair of sentences, texts or corpora,[27] it is often possible to judge whether they originate from a different domain (depending on the amount of text and the relative levels of difference, this may be more or less difficult of course). Much work implicitly or explicitly assumes that each corpus constitutes a single domain (one very notable exception being the Brown corpus (Francis 1979), which deliberately includes a wide range of genres). Thus, authors often mention 'the domain of newswire', 'the biomedical domain' and so on. We return to the question of quantitatively comparing corpora to evaluate the relative difference of their domains in Section 4.2.2.

Domain adaptation, then, is the task of applying linguistic methods tuned on one domain to some new domain. Here we are referring to the interaction of this with natural language parsing. It is relatively easy to motivate the need for domain adaptation here: if we wish to utilise the outputs of a given parser in some application, it is often the case that our target domain differs from that for which the parser was originally developed. These differences could include lexical item distribution, behaviour of these lexical items and even distribution of syntactic constructions. All of these differences can inhibit the effectiveness of the parse selection model, leading to a higher error rate in the target domain. This is not just a problem with the intrinsic parser evaluation; it will generally lead to more flow-on effects in downstream applications making use of the parser output. One example of this is noted by Vlachos (2010), who found that performance in a downstream task using the syntactic outputs was roughly correlated with the annotation effort for adapting to the new domain, and suggests that this effort is highly worthwhile, since the higher-quality adaptation can be reused for multiple tasks.

English-language parsers are often trained on the Penn Treebank, but unsurprisingly the domain of financial newswire text is not appropriate for many NLP applications. Gildea (2001) found that training a parser on the WSJ corpus rather than the Brown corpus[28] resulted in significantly worse performance over Brown corpus test data, reporting a 3.5% drop[29] in F-score over labelled constituents[30] from 84.1%. Gildea uses Model 1 of the Collins (1997) parser to measure changes in parser performance, but other work finds similar penalties with alternative parsers and domains as described below.

[27] A corpus is essentially a defined collection of linguistic data. In this context, we are referring to collections of sentences.

[28] This work only mentions comparing between 'corpora' possibly to avoid dealing with the ill-definedness of the concept of 'domain'.

[29] All percentage changes quoted in this section are absolute.

[30] The precision, recall and F-score figures in this work and that described below can be broadly grouped into two categories: (1) constituent-based evaluation, discussed further in Section 3.6.2, loosely or exactly following PARSEVAL (Black *et al.* 1991); and (2) dependency-based evaluation, which we explore in Section 3.6.3. These metrics produce different results, and even the relative changes should not be considered directly comparable. Results are often reported for only one of the two, however.

Some work goes further, also investigating strategies for avoiding these performance penalties when moving across domains. Roark and Bacchiani (2003) show that using a technique known as maximum *a posteriori* estimation on the productions in a probabilistic context-free grammar, it is possible to make more efficient use of in-domain and out-of-domain training data, giving labelled constituent F-score improvements of up to 2.5% over using only in-domain data when the amount of in-domain training is very limited (from a baseline of 80.5%), arguing that the conclusion of Gildea (2001) that out-of-domain data has very little value, was premature. Honnibal *et al.* (2009) found that the C&C parser (Clark and Curran 2007) trained on WSJ text gives a 4.3% lower F-score (based on CCG dependencies) when tested on Wikipedia data compared to held-out WSJ data (which had an F-score of 85.1%), but demonstrated that it was possible to reduce the penalty by training on automatically-parsed in-domain data which reduced this penalty to 3.8%. This is an example of a *self-training* strategy; this general concept and the work of Honnibal *et al.* are described in more detail in Section 2.6.1.

Plank and van Noord (2008) investigate domain adaptation of a parser trained on the Alpino Dutch treebank (van der Beek *et al.* 2002) using *auxiliary distributions*, by augmenting the model, which is trained initially on a small quantity of in-domain data, with a real-valued feature which takes the value of the negative logarithm of the conditional probability of the sentence according to the larger out-of-domain model. This approach achieves performance between 1% worse and 4% better than a model trained by simply combining the in-domain and out-of-domain data, improving the performance over a purely in-domain model by up to 1%, although over most test corpora there is only a small increase or decrease, indicating that integrating out-of-domain training data is difficult in this case. An alternative strategy of creating a model with only two features — the conditional probabilities from the in-domain and out-of-domain models — yields more modest improvements of around 0.6%, but more reliably.

For adapting WSJ-trained parsers into the biomedical domain, Clegg and Shepherd (2005) investigate the performance of three treebank parsers (Collins 1999; Charniak 2000; Bikel 2002) over the GENIA treebank (Tateisi and Tsujii 2004, as explored in Section 3.4.2), finding that labelled constituent-based F-scores are 8–9% lower than those obtained over WSJ data, and that these errors can be slightly ameliorated by combining the outputs of different parsers in various ways. Lease and Charniak (2005) observe that PARSEVAL F-score from the Charniak parser trained on the WSJ is lower by 13% over GENIA and 4% over the Brown corpus, compared with parsing in-domain data which gets an F-score of 89.5%. They show that using shallow domain-specific resources such as a domain-specific POS tagger and named entities from a medical thesaurus avoids some of the cross-domain training performance penalty, increasing the GENIA F-score by 3.3%. A somewhat similar conclusion is found by Rimell and Clark (2009) using the C&C parser, then mapping the parser output to the grammatical relations (somewhat like dependencies) of the

BioInfer corpus (Pyysalo *et al.* 2007) to calculate F-score. Using a domain-specific POS-tagger, and to a lesser extent a domain-tuned supertagger for the CCG lexical categories, improves F-score by 5.5% from the baseline of 76.0%. In the HPSG space, Hara *et al.* (2005), also working on the GENIA corpus, show that it is possible to augment a larger log-linear model trained on the WSJ with carefully selected features derived from a smaller in-domain treebank. They report a 1.6% improvement in constituent F-score compared to a baseline of 85.1% using a WSJ model only, and a 0.5% improvement over simply retraining a new model from the combined WSJ and GENIA training data, while greatly reducing the training time. In later work, Hara *et al.* (2007) show that simply retraining the lexical entry features (rather than the grammatical ones) could yield further improvements of around 2% over this method. McClosky and Charniak (2008) demonstrate a self-training strategy to adapt the Charniak parser to the biomedical domain which we describe in more detail in Section 2.6.1.

2.6.1 Self-training for Domain Adaptation

Co-training (Blum and Mitchell 1998) is a method of taking advantage of the wide availability of unlabelled data, compared to labelled data, which generally requires manual human input, and is thus scarce and expensive to create. As postulated originally, it is applicable when there are two different "views" (such as feature sets) of the training data which are conditionally-independent given the training label. Each view of the data is used to train a different learner, and the predictions of each of these learners are used to create training data for the other learner.

Self-training (Nigam and Ghani 2000) is similar, but only requires one learner, and the outputs of that learner are repurposed as training data for the learner itself; a survey is presented in Zhu and Goldberg (2009). Nigam and Ghani systematically compare self-training to co-training (and find that it performs less well on a text classification task), and may be the first to use the term. However given that this technique is so conceptually simple and unlabelled data is extremely abundant for most tasks, it is unsurprising that variants on this technique have (before and since) been applied to machine learning tasks under various names. Yarowsky (1995) applied a similar method denoted "bootstrapping" to the task of identifying cues for word-sense disambiguation,[31] while Charniak (1997), investigating natural-language parsing, applied a method described as "unsupervised learning" to slightly increase accuracy. Rosenberg *et al.* (2005), working in image processing, a field far outside NLP, also found that self-training could be usefully applied.

Self-training as applied to parsing (Charniak 1997; McClosky *et al.* 2006a) is, unsurprisingly, the process of using automatically-generated parser output as a source

[31] This is the task of identifying the sense of a word in context, such as the distinction between the sense of *bank* corresponding to the edge of a river, and the one corresponding to a financial institution

of parser training data. Some large corpus is parsed, and the top-ranked tree for each sentence is treated as if it had been manually annotated as a gold-standard tree. Throughout this thesis, we describe this best automatically-ranked tree as *pseudo-gold*, since it is treated as a gold-standard tree but was not annotated by a human. This data can then be used to train some component in the parsing pipeline. This is an attractive method as it effectively gives us arbitrarily large training corpora without requiring expensive annotation by expert treebankers, although the training data generally has less value than an equivalent manually-annotated corpus.

McClosky *et al.* (2006a, 2006b), McClosky and Charniak (2008) and Honnibal *et al.* (2009) all demonstrate some performance improvements from using self-training. In each case, the systems take advantage of a two-stage parsing process in the self-training procedure. McClosky *et al.* (2006a) use a two-stage reranking parser (Charniak and Johnson 2005), where a PCFG (similar to that of Charniak (1997) but with more effective features) is used for the first stage parsing, and the best 50 parses from there are then reranked according to a discriminative MaxEnt model. They found that using the best parse of the reranker as pseudo-gold could be used to retrain the first stage parser and thus provide a concomitant increase in accuracy of the complete reranking parser system. However, other combinations (e.g. using the parser output as training data for the parser) were found to be less useful and possibly mildly detrimental to parsing accuracy.

McClosky *et al.* (2006a) are actually focussed on improving performance over one domain (the Penn Treebank WSJ corpus) by augmenting the training data with self-trained data from a similar domain, so they are not strictly concerned with domain adaptation in this work. McClosky *et al.* (2006b) do investigate using self-training for domain adaptation to parse the Brown corpus, although interestingly this is most successful using a large newswire corpus (the same as in the 2006a paper) for self-training, while the Brown data itself is counterproductive for self-training. McClosky and Charniak (2008) apply the same technique to the task of domain adaptation in order to improve performance over the GENIA treebank without having access to a human-annotated domain-specific treebank, and in this case they do find that data from the target domain helps with self-training. The in-domain data is 270,000 sentences of randomly chosen Medline sentences, and when this was added to the hand-annotated out-of-domain data, the F-score over their GENIA development set increased from 82.6% to 84.1%. Subsequent work (McClosky *et al.* 2010; McClosky 2010), refines the domain adaptation process further, this time using the Charniak (2000) parser, by taking advantage of multiple training corpora (both manually annotated corpora and automatically parsed self-training corpora). A target domain is assumed to be a combination of the available source domains and a new parsing model is created by weighted combination of the models for the source domain. There are many different ways the various source models could be weighted; the weights which should produce optimal parsing accuracy are produced by a regression model which has been trained for this purpose. This strategy results in significant

F-score boost compared to other strategies, such as naive self-training without this technique of optimised selection of weights.

Honnibal *et al.* (2009) perform domain adaptation for the C&C CCG parser (Clark and Curran 2007). This parser has a supertagging stage as a pre-filter to the full-scale parsing for constraining the search space. They parse a subset of Wikipedia containing 185,000 sentences and use the best tree for each sentence as training data to retrain the supertagger. This gives a 0.8% improvement in F-score from 80.9% over using the faster *derivs* model of C&C, providing greater accuracy and faster parsing speed than using the slower and more complex *hybrid* model alone.

Other interesting work in this context comes from Sagae (2010), who primarily applied the Charniak (2000) parser, which does not include the reranking component of the Charniak and Johnson (2005) parser used by McClosky *et al.* (2006b). Sagae defines the term *simple self-training*, where the output of one component is used *directly* to train itself, rather than via some other component in the parsing pipeline — in this case, the generative parser. This work found that self-training could improve parser accuracy even without a reranking component — using 320k sentences of automatically parsed data from the Brown corpus as training data gave a 2% absolute improvement in F-score over the Brown corpus development set.

2.7 Translating Between Syntactic Formalisms

Human-annotated gold standard corpora are useful and often essential in a wide range of tasks in natural language processing. In particular, as we noted in Sections 2.4.1 and 2.4.3, human-annotated treebanks can be used both to induce grammars as well as to train statistical models for choosing between the possible parses of a given sentence, with both of these often conflated into a single task (Charniak 2000; Collins 1997). However treebank annotation requires extensive labour from specialist annotators, so large-scale treebanks are expensive to produce. For instance, even with extensive automated preprocessing and grammatical simplifications, the one million tokens of the Penn Treebank WSJ corpus correspond to roughly 1300 hours of specialist annotator time at the quoted rate of 750 words per hour (Marcus *et al.* 1993).

Understandably then, it is desirable to make maximal use of human-annotated resources. However, for a range of reasons, the assumptions and conventions associated with a particular resource may conflict with those required for a particular task. For example, the target task may use a framework based on an incompatible linguistic theory with different characteristics to that underlying the original treebank. Alternatively, there may be a need to maximise the amount of data available by combining heterogeneous treebanks. Another source of incompatibility could be incremental changes in a dynamic grammar that need to be reflected in an updated treebank.

In any of these cases, it seems plausible that even for very different treebanks, the underlying linguistic assumptions should have enough in common that there is some information encoded that would be compatible with both. Most syntactic formalisms have some notion of a hierarchical constituent structure, including HPSG as well as the other formalisms we mentioned in Section 2.2.4: GB (Chomsky 1982), LFG (Bresnan 2000) and LTAG (Joshi and Schabes 1997). It is not surprising that in addition to this, many linguistic resources also assume the existence of a constituent structure. The best-known of these is the Penn Treebank (Marcus *et al.* 1993), but there are many related treebanks such as the Penn Chinese Treebank (Xue *et al.* 2005) and the GENIA Treebank (Tateisi *et al.* 2005) (described in more detail in Section 3.4.2). The ability to reuse linguistic data does not absolutely require constituent-based annotations in any case — various methods can be used to convert to or from dependency-based representations,[32] such as in Forst (2003) and Niu *et al.* (2009) which we explore further below.

Some earlier work in this area is described by Wang *et al.* (1994), who present an algorithm for mapping between parse trees of a source grammar and a target grammar. With that algorithm, a given sentence is parsed using the target grammar to produce a parse forest which is generally ambiguous. From here, it selects the tree which most closely matches the source tree in terms of some constituent structure according to the defined scoring algorithm, which essentially minimises crossing brackets. They evaluate this work on trees parsed with two different but related versions of a grammar, although there is no specific reason why it could not be applied to unrelated grammars, except that in that case there would be more chance of mismatches resulting in incomplete disambiguation, requiring arbitrarily choosing between parses.

In Section 2.4.1, we outlined the work of Xia (1999), Hockenmaier and Steedman (2002) and Miyao *et al.* (2004). All of this work is a kind of translation between syntactic formalisms. Each approach uses the grammar implicitly encoded in a particular treebank to create a treebank for a completely different linguistic framework, and uses this to induce a grammar in that framework.

Diverging from this grammar conversion approach, but particularly relevant to the work described in Chapter 5, Riezler *et al.* (2002) take a *pre-existing* grammar and use the grammatical constraints implied by the PTB to build a discriminative estimation model for choosing between parses. In this case, the pre-existing grammar is a English-language grammar in the LFG framework from the ParGram project (Butt *et al.* 2002). The process uses a trimmed-down version of the PTB, discarding POS tags and labelled brackets they deem to be uninformative for LFG. By requiring that particular labels correspond to particular elements in the c-structure and f-

[32] *Dependencies* relate items in sentences using links between nodes corresponding to words, rather than with constituent structure. These are used by some annotated corpora such as DepBank (King *et al.* 2003) and the Prague Dependency Treebank (Hajič 2006) instead of PTB-like trees, and are the native output of some parsers such as MaltParser (Nivre *et al.* 2006).

structure,[33] and comparing each tree with corresponding set of LFG parse trees for the same sentence, they are able to select the set of parse trees that are consistent with the PTB annotations. The comparatively shallow PTB annotations are rarely able to fully disambiguate the more complex LFG trees, but they outline a procedure for calculating discriminative estimation parameters that is able to take advantage of the partially-disambiguated parse forest. The testing data is semi-automatically created by post-correction of LFG parses of 700 PTB sentences, and they achieve a 3-5% improvement in dependency-based F-score (including some parses that were created heuristically from fragment analyses for sentences outside the coverage of the grammar).

Forst (2003) also worked with LFG, but for the German language. The grammatical relations from the TIGER corpus (Brants *et al.* 2004) are extracted and converted into a Prolog representation. Following this, transfer rules such as those which might be seen in a rule-based machine translation system are used to account for the differences between the assumptions underlying the TIGER representation and the LFG representation. The primary aim was to create a test corpus of partially underspecified LFG f-structures, although the work notes that the f-structures created could also be used as training data.

To make maximal amounts of training data available, it may also be desirable to convert between two treebanks encoded in different formalisms. The previously mentioned work used constituency treebanks, which store phrase structure trees, but treebanks can also be encoded as grammatical dependencies, and there is not necessarily a one-to-one mapping between them (there are also likely to be different underlying grammatical assumptions causing this). Niu *et al.* (2009) present an approach for converting from a dependency treebank to a phrase structure treebank, by parsing the sentences from the dependency corpus (the Chinese Dependency Treebank, or CDT) with a phrase structure parser, and scoring the outputs by dependency F-score against the gold-standard from the CDT, selecting the top-ranked standard as a new gold-standard best parse. Zhu *et al.* (2010) use a different method to combine information from phrase structure treebanks with different conventions, training the same parsing algorithm once on each treebank and 'co-decoding', or exchanging information between the trained parsers to help rank each tree. This is based on the intuition that even for treebanks based on different annotation conventions, there should be a large amount of commonality in the structures in each, so the derived models may be somewhat informative even for an incompatible treebank. In both cases, modest improvements are achieved over a benchmark parser.

Thus, much research has shown that some useful information can be extracted from superficially incompatible parse trees, but shallower human-annotated resources can also provide valuable information for parse forest pruning. For example, gold-

[33] Recall from Section 2.4.2 that c-structure describes surface syntactic form while f-structure describes the function.

standard POS tags may not remove all sources of ambiguity, but they can reduce ambiguity in the parse forest. This possibility is explored by Tanaka *et al.* (2005), who manually created a Japanese treebank (Bond *et al.* 2004) by selecting the best parses from those offered as candidates from JaCy, the HPSG grammar of Japanese mentioned in Section 2.4.3. The annotators reject or affirm *discriminants* to select the best tree, as is described in more detail in Section 3.3.4. Their data was already human-annotated with POS tags, which they used to constrain the parse forest, requiring on average 19.5% fewer decisions and 15% less time per tree.

2.8 Information Extraction

Information Extraction (IE), as the name suggests, is about extracting the information contained within documents. IE, as covered by Jackson and Moulinier (2007), is generally concerned with converting some of the unstructured human readable text in a set of documents into a structured format derived from the semantics of the text, allowing the information to be more directly utilised computationally. For example, this structured data can then be used for sophisticated querying over the documents or flagging a document as important, to compare two documents for similarity, or to extract certain key facts from the text. IE is particularly important when there is a large volume of relevant text, and when this text is heavily laden with information which may be of interest. In such cases the structured output of an IE system can provide useful insights into the text without requiring humans to perform laborious close readings of the complete body of text.

2.8.1 Biomedical Information Extraction Overview

The text contained in published biomedical research papers fulfils both of these criteria for the potential utility of IE techniques. The prose of the papers obviously comprises a large body of very valuable information in a mostly unstructured format, and the amount of such papers available is large, and rapidly increasing. The last several decades has seen a huge expansion in publication of biomedical research papers, not only in the total volume of work available, but the rate at which new work is published. The primary search interface for biomedical research publications, PubMed,[34] lists 20.6 million articles with publication dates of 2010 or earlier. There is unsurprisingly an ever-increasing total volume of papers indexed, but even the rate at which papers are added is increasing. Figure 2.11 shows the number of publications available in the online index with publication dates within each calendar year from 1967 to 2010. With the slight exceptions of 1980 and 1997, each successive calendar year sees more publications added than in the previous year. In 2010, there were 918,000 articles added, corresponding to 2,500 articles per day.

[34] http://www.ncbi.nlm.nih.gov/pubmed

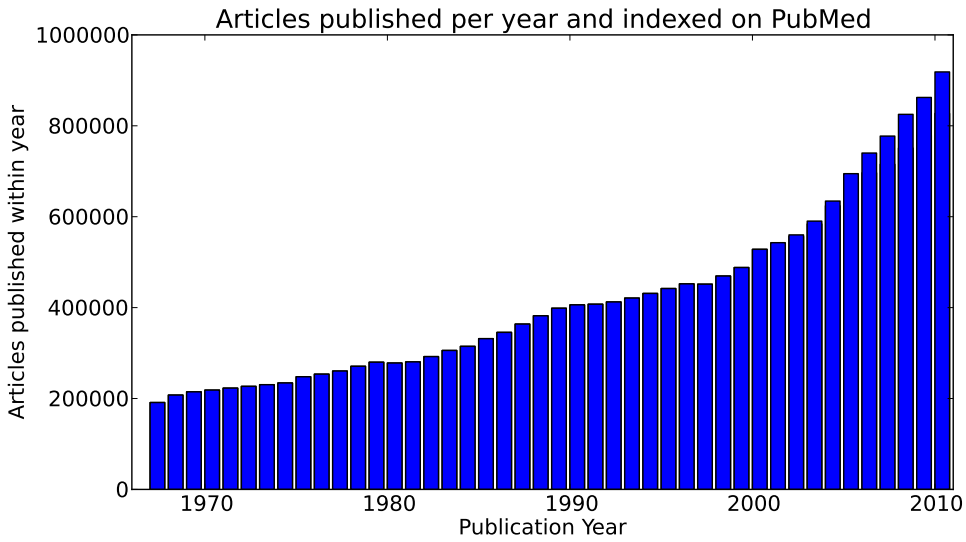


Figure 2.11: Number of articles published per year and indexed on PubMed. Each bar represents the publication output for the particular year, not the cumulative total. The numbers are the results of a search query on PubMed (`http://www.ncbi.nlm.nih.gov/pubmed`) by publication date — e.g. '`2010/01/01:2010/12/31[dp]`' for the 2010 calendar year

While researchers in the broad field would not have to peruse all of these to determine if they are relevant to their particular subfield, there is still demand for more fine-grained methods for locating relevant research, beyond the standard information-retrieval techniques which are often used. All new articles and most old articles have abstracts available, and an increasing proportion have full-text available in open-access journals. This means there is a large body of information embodied in free natural language text to which it should be possible to apply natural language processing techniques.

It is unsurprising then, that there is a growing body of NLP research concerned with processing this academic language from biomedical researchers (Verspoor *et al.* 2006). Biomedical NLP can be concerned with adapting standard upstream processing techniques to work in the biomedical domain — for example, part-of-speech tagging (Tsuruoka *et al.* 2005; Buyko *et al.* 2006), treebank construction (Kulick *et al.* 2004; Tateisi *et al.* 2005), parsing (Lease and Charniak 2005; McClosky and Charniak 2008; Rimell and Clark 2009; Buyko *et al.* 2006) and named-entity recognition (Kim *et al.* 2004; Zhou and Su 2004; Hirschman *et al.* 2005; McDonald and Pereira 2005; Sasaki *et al.* 2008).

However, much work also takes place on downstream tasks — which are more likely to be consumers of the outputs from the tasks mentioned above. There is, unsurprisingly, research within the intersection of biomedical NLP and information extraction. However, in biomedical IE, the nature of the domain affects the tasks that are attempted by researchers. For example, there is a substantial body of work on *protein-protein interaction* (Sætre *et al.* 2007; Zhou and He 2008; Airola *et al.* 2008; Miyao *et al.* 2009) — determining from free text where proteins are mentioned, and whether they interact, as well as the nature of that interaction. Much of the research is driven by shared tasks, where task organisers create a data set and participants all apply their methods to that data. For example, the first BioCreAtIvE challenge (Hirschman *et al.* 2005) was designed to assess the state-of-the-art from some information extraction subtasks over biological texts. The first subtask was to identify mentions of genes or proteins in abstracts from MEDLINE[35] — much like a specific kind of named-entity recognition — and optionally map them to normalised gene names. The second subtask was to find evidence within full-text articles to support annotations in a gene ontology. BioCreative II, (Krallinger *et al.* 2008) and BioCreative III (Arighi *et al.* 2011), which were subsequent iterations of the BioCreative challenge, added protein-protein interaction tasks.

2.8.2 The BioNLP 2009 Shared Task

The BioNLP 2009 shared task (BN09ST: Kim *et al.* (2009)) is a shared task in biomedical IE which is of particular relevance later in this work. The BN09ST attempts to target more nuanced entities and interactions than those of the BioCreAtIvE challenge, ultimately to assist in the construction of curated biomedical databases. The primary entities are also proteins, but the training and test data has gold-standard annotations for these proteins, to focus attention on other downstream aspects of the task. The primary subtask, denoted Task 1, is to identify salient biomedical events involving these proteins, and possibly other entities, from a set of eight event types, described in more detail in Section 6.2.

Task 2 of the BN09ST relates to detecting secondary arguments of these events, such as locations and binding sites, but is not relevant here. Of particular interest is Task 3, which involves detection of modification of the events annotated in Task 1. The prose discussing a particular event may indicate that it is unclear whether the event occurred (SPECULATION) or that the particular event in fact did not occur (NEGATION). The goal in Task 3 is to determine which events are subject to such modification.

For a concrete example of Tasks 1 and 3, consider the following sample sentence, which already has the initial protein named-entity annotations (these are provided with the shared task data) as shown in square brackets.

[35] The database of citations which comprises the majority of research papers searchable on PubMed

(2.35) *⌈*$_{\text{protein}}$ *TRADD ⌋*$_1$ *was the only protein that interacted with wild-type ⌈*$_{\text{protein}}$ *TES2 ⌋*$_2$ *and not with isoleucine-mutated ⌈*$_{\text{protein}}$ *TES2 ⌋*$_3$.

The aim for Task 1 is to identify the trigger words in the sentence and the corresponding events and to link those events to participating entities, while Task 3 requires detecting modification of these events. The gold-standard target trigger word annotations, which task participants must attempt to reproduce, give a single trigger-word for this particular sentence, as shown:

(2.36) *⌈*$_{\text{protein}}$ *TRADD ⌋*$_1$ *was the only protein that ⌈*$_{\text{trigger}}$ ***interacted*** *⌋*$_4$ *with wild-type ⌈*$_{\text{protein}}$ *TES2 ⌋*$_2$ *and not with isoleucine-mutated ⌈*$_{\text{protein}}$ *TES2 ⌋*$_3$.

The gold standard also stipulates that these triggers are linked to the following events, and also that there is event modification occurring:

1. Event evt_1
   TYPE = BINDING
   TRIGGER = [$_{\text{trigger}}$ *interacted*]$_4$
   THEME$_1$ = [$_{\text{protein}}$ *TRADD*]$_1$
   THEME$_2$ = [$_{\text{protein}}$ *TES2*]$_2$

2. Event evt_2
   TYPE = BINDING
   TRIGGER = [$_{\text{trigger}}$ *interacted*]$_4$
   THEME$_1$ = [$_{\text{protein}}$ *TRADD*]$_1$
   THEME$_2$ = [$_{\text{protein}}$ *TES2*]$_3$

3. Modification mod_1
   TYPE = NEGATION
   THEME = evt_2

In other words, the sentence refers to two salient biological events — evt_1 and evt_2, which both refer to a binding event involving the *TRADD* and *TES2* proteins. Both events have the same trigger word, and [$_{\text{protein}}$ *TRADD*]$_1$ as a theme, but there are two different second themes corresponding to the different *TES2* entities in the sentence. These comprise the gold standard Task 1 annotations. There are no examples shown here of events as arguments to other events, but as noted above, this is a possibility for the regulation events.

For Task 3 in this example, evt_2 is negated, as shown by mod_1. This should be apparent to a human reader of the text, as the second *TES2* mention is part

of a prepositional phrase which is negated by the *not* token in the sentence. The other type of modification mentioned above for Task 3 is SPECULATION, which is not demonstrated in this example. This would occur where the prose indicates that the event was hedged or speculative, such as in *analysis of IkappaBalpha phosphorylation*, where the usage of the word *analysis* suggests that it is not clear whether *IkappaBalpha phosphorylation* occurred or not.

Task 1 Systems

The best-performing system on Task 1 came from Björne *et al.* (2009). For trigger detection, they treated the task as a token labelling problem, where each token is evaluated trigger class to which it belongs. This must be either a class corresponding to one of the event types, or the negative class for the majority of words which are not event triggers. The trigger classification uses a number of token-based features including capitalisation, character n-grams,[36] word stem, and presence of the token in a list of known trigger tokens from the training data. There are also frequency-based features based on bag-of-words[37] for the sentence and number of named-entities, and dependency features up to depth three based on dependency types as well as the same token-based features for each link in the chain. The features are supplied to the multi-class support vector machine (SVM) implementation described by Joachims *et al.* (2009), and the output is post-corrected to explicitly tune the precision/recall tradeoff.

This gives an event and type associated with each postulated trigger word. From here, each possible event is considered as part of a graph linked to all protein entities and other event triggers, and each edge in the graph is classified as either THEME or CAUSE, meaning the linked node is an argument of the event, or as the negative class indicating no relationship. The classification is performed with another multi-class SVM, and makes use of syntactic dependencies — specifically the shortest undirected path through the dependency graph between the head tokens corresponding to the protein entity or trigger. The classification features use token attributes consisting of token text, part-of-speech, and entity or event class. The features include component features for individual tokens or edges, n-gram features of 2–4 tokens based on target tokens and immediate dependency neighbours, both of which also merge the token attributes, and semantic node features based on the terminal nodes of each possible event-argument link, incorporating the event/entity class. This system obtained precision/recall/F-score of 58.48/46.73/51.95% over the test set using approximate recursive match evaluation, the primary metric for the task as mentioned above.

[36] n-grams are linear sequences of words or, in this case, characters from the text

[37] A *bag-of-words* is simply a collection of tokens from a sentence or document with the original word-order discarded

A follow-up study was presented by Vlachos (2010), which for the trigger-detection stage emulated the approach of Vlachos *et al.* (2009) using a dictionary of lemmas[38] extracted from the training data. This is achieved by lemmatising the triggers in the training data and discarding singleton terms, then removing stopwords from the sequence of trigger tokens and associating each single-lemma trigger with the event class it most often corresponds to, or to multiple classes when it is consistently associated with them. There are also some optimisations to handle *light trigger* lemmas which occur in both single-token and multi-token triggers, and map to different event classes in each case. The dictionary is used by simply examining each lemmatised training token and postulating a linked event of the associated class, with some post-processing for the light trigger lemmas.

There are two approaches for the argument identification stage. The first emulates the rule-based approach of Vlachos *et al.* (2009), which used the grammatical relations (GRs) from RASP (Briscoe *et al.* 2006), and matched GR paths connecting trigger words to potential arguments against manually-created GR patterns to identify the arguments. Vlachos (2010) replaces the syntactic parser inputs from RASP with those of the Charniak and Johnson (2005) parser as supplied to the participants in the original shared task — namely, parsed using the domain-adapted parsing model of McClosky and Charniak (2008) and converted into Stanford dependency format. There is a set of six rules relating the syntactic relationship between the trigger token, and the argument token, which can either be another trigger token or a protein named entity — for example, an argument token as the subject of a trigger token, as in $[_{\text{protein}}$ *Stat-1* $]_1$ $[_{\text{trigger}}$ *expresses* $]_2$. This approach achieves an F-score of 35.39% over the test data using the approximate recursive match metric.

The second approach applies machine-learning to the argument identification procedure as this was a source of weakness in the rule-based approach. Somewhat similarly to the approach of Björne *et al.* (2009) described above, this considers each pairing of a detected trigger token with a candidate argument (either a named entity or another trigger, as a proxy for its event). In this case however, there are two SVM classifiers, one binary classifier to classify each candidate argument as THEME or non-THEME the simple event types and BINDING (which have only THEME arguments) and a ternary classifier for REGULATION events and their subtypes, to classify each candidate as a THEME, a CAUSE or neither.

The feature set is somewhat simpler than Björne *et al.*. For each trigger-argument pair, the dependency path in either direction is extracted (ignoring those items with dependency paths longer than four in the training data) along with the trigger token, the trigger event type and the argument type. At classification time, instances with unseen dependency paths are assigned to the negative class. In the best-performing configuration, the dependencies from the McClosky and Charniak and Clark and Curran (2007) parsers are used to generate two parallel versions of the dependency

[38] Broadly, a *lemma* is a base wordform without inflectional suffixes

path features, as it was hypothesised that the different parsing algorithms would have different strengths and weaknesses, which was supported by experiments on the development set. Over the test set, this system achieved a precision/recall/F-score of 56.76/41.42/49.35% using the approximate recursive match metric.

Another approach to Task 1 is presented in MacKinlay *et al.* (2009); the Task 3 methods from this work are described and developed further in Chapter 6, which makes use of this output although the Task 1 method itself is not a contribution of this thesis. The most successful configuration used a combination of two different methods. One was a lookup-based method using a dictionary of trigger words automatically extracted from the training data, where tokens which occurred more than some threshold are inserted into a dictionary, associated with the event type most frequently observed in the training data. This method achieved higher recall than precision. The second approach used Conditional Random Fields (CRFs: Lafferty *et al.* (2001)), a machine learning algorithm which generalises maximum entropy methods and can label entire sequences simultaneously. The features for the submitted results came from a local context window of 3–4 tokens and used wordforms, lemmas, POS-tags, sentence chunks and protein named entities. The CRF approach has comparatively low recall. In an attempt to combine the advantages of both, the system submitted heuristically chose between the lookup-based method and the CRF-based method, giving a precision/recall/F-score of 17.44/39.99/24.29% over the test set using the standard metric.

Task 3 Systems

Several different approaches were applied to Task 3. Comparing systems over Task 3 in isolation is difficult, since most systems used a pipeline architecture, as it was necessary to identify events before making hypotheses about modification of them. The evaluation metric means that sub-optimal performance in Task 1 has a detrimental effect on both precision and recall in Task 3, as a penalty is applied for detecting modifications of events which do not correspond to anything in the gold standard but were postulated by the event detection module of Task 1.

The system of MacKinlay *et al.* (2009) which used machine learning and semantic analyses is explained in detail and extended in Chapter 6, but the best-performing approach to Task 3 in the original shared task (noting the above caveat) was the rule-based approach of Kilicoglu and Bergler (2009). This used a dependency representation of the sentence (De Marneffe *et al.* 2006), from the output of the Stanford Lexicalised Parser (Klein and Manning 2003).

The speculation detection module is a refinement of the one outlined in Kilicoglu and Bergler (2008), which was aimed at categorising a whole sentence as speculative or not, rather than an individual event as in the shared task and used a completely unrelated data set. In this original module, a large portion of the hedge detection is based on a set of surface lexical cues in several different classes:

- Modal auxiliaries: *may*, *should*, *might*, ...
- Epistemic verbs: *suggest*, *indicate*, *appear*, *infer*, *deduce*, ...
- Epistemic adjectives: *likely*, *possible*, ...
- Epistemic adverbs: *probably*, *perhaps*, ...
- Epistemic nouns: *suggestion*, *possibility*, ...

The starting point was a seed list of 63 hedging cues including those listed above. This was expanded semi-automatically in two stages. The first used WordNet (Miller 1995), and each seed term was expanded to include those in the synsets of the seed term, ignoring words that did not appear in the training data, and in the case of verbs, those synonyms that did not subcategorise for a *that*-complement. The verbs and adjectives from this expanded set are then subject to further expansion into their nominalised forms, as nominalisations are frequent in formal academic biomedical text. This is performed using the UMLS SPECIALIST Lexicon (McCray *et al.* 1994), which includes morphological annotations for the entries in biomedical and general English. Finally, there was a set of "unhedging" cues such as *demonstrate* and *prove*, which indicate certainty but when negated are also strong indicators of hedging. This resulted in a dictionary of 190 hedging cues. This dictionary of terms was then used to construct dependency patterns which could be applied over the dependency output of the Stanford Parser, with a heuristically assigned hedging strength associated with each pattern. Some of the patterns are:

- [EPISTEMIC VERB] *to(inf)* [VERB]
- [EPISTEMIC NOUN] *that(comp)*
- *not* [UNHEDGING VERB]

In the 2009 shared task submission, there were a number of differences as noted above. Firstly classification is at the event level, not at the sentence level, so the dependencies they examined were those on the dependency paths which included the event trigger, although it is not clear whether the methodology was altered any more than this from that described above. There were also changes to the cues and patterns used. Firstly the modal verbs were found to not influence speculation so were removed from the cue set, and a new class of verbs was added, which they denote *active cognition* verbs, such as *analyze*, *study* and *examine*. This was used in the new syntactic pattern they added which looked for a dependency pattern between any verbs from this set (or their nominalisations) and an event trigger, and marked the event as speculative if, for example, the trigger is a direct object of the verb.

Negation detection similarly checked for the presence of particular cues and dependency paths between these cues and the trigger terms, corresponding to patterns such as the following:

- [*lack* | *absence*] *of(prep)* [NOUN TRIGGER]
- [*inability* | *failure*] *to(inf)* [VERB TRIGGER]

There are also dependency types which do not require particular lexical cues — specifically an event trigger involved in a NEG dependency (e.g. constructions with *not*), or event arguments or triggers involved in a CONJ_NEGCC (seen with conjunctions such as *but not*).

As is often the case with rule-based systems, this approach favoured precision over recall. In addition, we noted above that Task 1 usually has a strong effect on Task 3. The Task 1 system here had a precision/recall/F-score of 61.59/34.98/44.62% over the test set, and over Task 3 they achieved scores of 50.75/14.98/23.13% for negation and 50.72/16.83/25.27% for speculation.

A much simpler but somewhat effective rule-based approach to Task 3 was presented by Van Landeghem *et al.* (2009). Rather than using any syntactic information, they use pattern matching and some manually-constructed dictionaries. For speculation, they compiled a list of expressions denoting uncertainty, such as *we have examined whether* and looked for these expressions within 60 characters on either side of the trigger, as well as expressions indicating a hypothesis is being postulated to explain the results of the experiment, such as *might* or *appear to*, then searched for these in the 20 characters preceding the trigger. For negation, they looked for negation constructs such as *no* or *failure to* immediately before the trigger, intrinsically negative triggers such as *non-expressing*, and semantically negative conjunctions such as *but not* immediately before a protein argument of an event. From a Task 1 precision/recall/F-score of 51.55/33.41/40.54%, they obtained Task 3 results of 45.10/10.57/17.13% for negation and 15.79/8.65/11.18% for speculation.

2.8.3 The BioNLP 2011 Shared Task

The BioNLP 2011 shared task (BN11ST: Kim *et al.* (2011)) is a follow-up task closely-related to the BN09ST. The task definition for the "GENIA Event" (GE) task of the BN11ST is identical to that of the BN09ST, while the data supplied is a superset. In addition to the same set of annotated abstracts, there is also a set of completely-annotated fulltext articles, in order to evaluate how well techniques developed on abstracts generalise to the bodies of articles. There are 14 annotated full-text articles of roughly 6000 words each, with five of these in the training set, five in the development set and four in the test set. There are also several other component tasks introduced in the BN11ST but we do not cover them here.

For the GE task, some submissions were based heavily on submissions to the BN09ST, with some adjustments. The submission of Björne and Salakoski (2011) uses largely the same system as Björne *et al.* (2009) for GE Task 1, but they generalise it to handle Tasks 2 and 3 as well. The Task 3 detection of speculation and negation

uses two independent classifiers, based on largely the same features as their trigger detection system, with the addition of a list of speculation-related words determined on the basis of the BN09ST data. Their system obtained an F-score of 53.13% for Task 1 over the test data of BN09ST, a small improvement from the 2009 submission which had a score of 51.95%. Over the full 2011 test set, their Task 1 F-score was 53.30% and their task 3 F-score was 26.86%.

The only other system in GE Task 3 was that of Kilicoglu and Bergler (2011). This was also based heavily on an earlier BN09ST submission — in this case the rule-based system of Kilicoglu and Bergler (2009), which we described in the previous section. The Task 3 classifier was very similar, but with an expanded list of speculation and negation cues, as well as allowing affixal negation. They achieved F-scores over the test data of 50.32% for Task 1 and 26.83% for Task 3.

Other systems showed slightly more divergence from their ancestors. The system of Riedel and McCallum (2011), which was one of the strongest performers, uses a similar method of representing events and arguments as a graph as Riedel *et al.* (2009) (and indeed Björne *et al.* (2009)), with some modifications to explicitly record the relationship of proteins with a particular kind of event known as 'binding', which may involve one or more proteins. However the learning model differs somewhat. The newer work uses a *joint model* composed of three submodels corresponding to different relationships between events and their arguments and uses a technique known as *dual decomposition* (Rush *et al.* 2010) to simultaneously make inferences using all three submodels. This means the event triggers and their arguments can be inferred in a single step. This system obtained F-scores of 55.2% for Task 1 and 51.0% for Task 2.

McClosky *et al.* (2011) describe a system which has no predecessor in the original BN09ST. The first stage is a trigger detection system using features derived from a constituent parse tree similar to those of Björne *et al.* (2009). Each token is classified as an event type or *none* using a logistic regression model. Following this, the linking of events with arguments is treated as a form of dependency parsing. Using an off the shelf dependency parser, only the relevant entities and event triggers are parsed then converted into event structures. Finally, these structures are reranked based on features derived from the structures themselves. Over GE Task 1, they obtained an F-score of 50.0%.

2.8.4 The CoNLL 2010 Shared Task

The CoNLL 2010 Shared Task (Farkas *et al.* 2010) is also concerned with detection of speculative language (or *hedging*) in biomedical text (as well as other domains), but unlike in the BioNLP shared task described above, detection of speculation is the sole focus — there is no component for detecting entities or events, or negation-type modifications of these. The first component of the task, Task 1, required participants to identify sentences which contained uncertain or unreliable information — i.e. to classify each sentence in the test set as either certain or uncertain. There were two

domains for Task 1 — biomedical abstracts and full-text articles taken from the BioScope corpus (Vincze *et al.* 2008), as well as a set of annotated Wikipedia sentences, although we do not cover work using the latter set in detail here. The cues indicating hedging were annotated in the sentences, although correctly identifying these was not part of the Task 1 evaluation.

Task 2 required participants to correctly identify cue phrases indicating hedging, as well as the scope of the hedge phrase — i.e. the span within the sentence which falls within the scope of the speculation. This subtask only targeted biomedical data. In the example below taken from the sample data the cue, *suggesting*, is shown in bold, and the scope of the span which is hedged is bracketed:

- *In addition, treatment of these nuclear extracts with sodium deoxycholate restored their ability to form the heterodimer,* $[_{\text{hedge}}$ ***suggesting*** *the presence of an inhibitor of NF-kappa B activity]*

While there is the additional requirement of needing to identify the hedge cue and the exact scope of the hedged phrase, there is clearly some commonality between this and Task 3 of the BioNLP shared task — in the latter, the participants were probably implicitly identifying hedge cues and evaluating whether the event triggers fell within their scope, although it is a more forgiving evaluation metric for two reasons. Firstly, scope cues do not need to be explicitly identified. Secondly, it is not necessary to determine the exact boundaries of the hedging scope — it only matters whether it covers the point where the trigger occurs.

One of the best-performing systems for Task 2, and most relevant here, is the work of Velldal *et al.* (2010). Their approach uses some syntactic information for tasks 1 and 2. Some of this comes from lexical-function grammar (LFG) parses from the XLE system (Crouch *et al.* 2008) utilising the associated English grammar (Butt *et al.* 2002), which are then converted to dependency analyses. They also use dependency parses from MaltParser (Nivre *et al.* 2006), which were enhanced using the XLE dependency analyses.

To identify hedge cues (and correspondingly hedged sentences) in Task 1, they supply syntactic and word-based features to a maximum entropy classifier. The feature set is defined relative to the cue word and includes n-grams over surface forms and base forms in a window of up to three tokens left and right, POS of the word according to the GENIA tagger (Tsuruoka *et al.* 2005) and three other features derived from the XLE syntactic analysis of the sentence, denoted 'coord', 'coordLevel' and 'subcat'. The 'coord' feature indicates whether the word is a conjunction, the 'coordLevel' feature indicates the syntactic category of the co-ordination (such as NP or VP), and the 'subcat' feature refers to the subcategorisation information from XLE about a verb, such as whether it is modal. They also experimented with a range of other features based on syntactic dependencies, such as the dependency path to the root, although these were not promising enough over the training data to be included

in the final feature set. The trained learner was applied to the candidate cue words, and those sentences with at least one token labelled as a hedge cue were classified as uncertain. The score over Task 1 at the sentence level was 85.48/84.94/85.21% for precision/recall/F-score, placing this system fourth overall. For detecting cue words themselves (sometimes spanning multiple tokens), the precision/recall/F-score was 81.20/76.31/78.68%. This was not part of the official evaluation but is indirectly relevant for Task 2.

In order to identify the scope of the hedging for Task 2, they start by assuming the scope extended from the detected cue to the end of the sentence, and use a set of heuristics to refine this, based on the two sets of dependency analyses and conditioned on the POS of the cue word. For example, if the cue is POS-tagged as a conjunction, the scope of the hedging is over the conjoined elements, and if the cue is an attributive adjective, the hedging scope is set to cover the head of the corresponding noun phrase and its descendants. There is a set of 8 such rules in total, conditioned on POS, as well as some special handling of multi-word cues, applying to parts-of-speech which frequently occur as cue words: conjunctions, prepositions, adjectives, verbs (with special handling of modals) and adverbs. This strategy gave them a precision, recall and F-score of 56.7/54.0/55.3% over the test data, placing them third overall in the shared task for Task 2.

2.9 Summary

In this chapter we have outlined the general background knowledge required for comprehension of this thesis. We gave a brief exposition of natural language syntax, focussing in particular on the Head-driven Phrase Structure Grammar formalism. This underlies the computational grammar of English which is the basis of the experiments in this thesis. Modern natural language processing seldom takes place without some usage of statistical techniques such as those of supervised machine learning, so for this reason we also described this paradigm, and particularly learning techniques based on Maximum Entropy methods which we use heavily later in this work. One of the important places in which machine learning methods are used is in parsing natural language to automatically apply syntactic analyses. We gave a brief overview on parsing and the mainstream approach of inducing grammars from pre-annotated treebanks of syntactic analyses, and contrasted this with the approach used in this thesis of analysing sentences with a pre-existing hand-crafted grammar, then ranking those analyses. This parsing process also often uses external supporting tools such as part-of-speech taggers, which we also described.

The remainder of the chapter was concerned with various techniques which are relevant for certain sections of this thesis. Domain adaptation for parsing attempts to avoid the performance penalty when we have no in-domain training data, while translating between syntactic formalisms is used when our training data is not directly

compatible with the target data format. We also described information extraction, focussing particularly on that part which is concerned with detection of modification of events described in biomedical research abstracts.

In Chapter 3, we will move beyond the general-purpose background information to described the specific resources we use throughout the thesis.

Chapter 3

Resources

In Chapter 2, we outlined general NLP knowledge which is useful for comprehending this thesis. In this chapter, we give an in-depth discussion of the tools and resources such as grammars, treebanks, software and semantic formalisms which are directly used in this thesis.

3.1 Grammars

3.1.1 The English Resource Grammar

The nature of HPSG makes it well-suited to computational language processing. We have already discussed in Section 2.4.3 that there are several hand-crafted HPSG grammar implementations as part of the DELPH-IN suite of grammars and associated software, and that one of the most thoroughly developed is the ERG (Copestake and Flickinger 2000; Flickinger 2000), a hand-built deep precision grammar of English.

The grammar was initially developed to be the deep parsing component within the Verbmobil project (Wahlster 2000), a speech-to-speech machine translation project including English, German and Japanese as languages, and as such was targeted at the constructions found in spoken language, with a bias towards the vocabulary in the Verbmobil test corpus. More recently, it was developed for the LOGON project[1] (Lønning *et al.* 2004), another machine translation effort, in this case targeted at translating from Norwegian to English, using hand-built grammars for each language. The target text for this project for the ERG was English translations of texts on Norwegian hiking (discussed further in Section 3.4.1), so there is also some custom-tuning of the ERG to this domain. At the time of writing of this thesis, it is still under active development, so we specify the grammar version where this is important.

The grammar is implemented primarily as a set of rules and lexical entries in the TDL language (Krieger and Schäfer 1994), a declarative language for specifying

[1] http://www.emmtee.net

typed feature structures particularly targeted at enabling HPSG implementations. It also includes various customisations for tasks such as preprocessing. The rules are reasonably closely modelled on the core HPSG formalism of Pollard and Sag (1994) and its subsequent development, and in fact influenced that subsequent development (Flickinger 2000). As such, it includes familiar looking HPSG rules such as Head-Complement rules and Specifier-Head rules, while accounting for a large range of linguistic phenomena within the lexicon and unsurprisingly using inheritance to avoid redundancy.

The grammar is a deep grammar according to the definition of Cahill *et al.* (2008), meaning that one of its characteristics is providing a semantic representations corresponding to the input string. The ERG produces these fine-grained semantic representations by storing information in feature structures as the derivation tree is created, which we explain in more detail in Section 3.5.4. The semantic formalism used for this is Minimal Recursion Semantics (MRS: Copestake *et al.* 2005), which is covered in more depth in Section 3.5.1. With appropriate external machinery, it is also possible to apply the inverse operation, and use the grammar to *generate* — that is, create textual strings from an input MRS, which is discussed in Section 3.5.5.

The ERG includes a fairly large collection of *lexical entries* making up its lexicon. Each of these inherits from a *lexical type* which could be described as a "word class", although these are much finer-grained than parts-of-speech, as they include many subtle lexical attributes such as accurate valence information for nouns, verbs, adjectives and prepositions. These lexical types are arranged in a hierarchy to maximally reuse the specifications and reflect linguistic commonality. The '1010' version of the ERG has 1003 leaf lexical types and 35435 lexical entries. More detailed statistics categorised by part-of-speech are shown in Table 3.1.[2]

The lexicon operates in conjunction with rules of the grammar. These are also arranged in a hierarchy, and are divided into *lexical rules* and *construction rules*. The syntactic construction rules reflect the various HPSG rule schemata including those we have covered such as the Head-Complement rule. Like the rules we demonstrated in Section 2.2.4, each takes some sequence of feature structures (the children) and converts it into a single feature structure (the parent). They are at most binary branching — i.e. there are at most two children for each parent, and in some cases they are unary branching.

The lexical rules, meanwhile, are divided into inflectional rules, derivational rules and punctuation rules. The inflectional and derivational rules handle the

[2]Interestingly, the open classes show fairly different numbers of entries per lexical type. There are around 60 entries per type for adjectives and nouns, but only around 25 per type for verbs. This is probably mostly a reflection of a greater syntactic diversity of verbs in English — to explain phenomena such as verb valency variation satisfactorily, we need greater granularity in the lexical types, and thus see fewer lexical entries in each one.

	Lexical Types	Lexical Entries
All	1003	35435
Adjective	104	5706
Adverb	84	2085
Complementizer	14	18
Conjunction	60	77
Determiner	50	105
Noun	265	18376
Preposition	60	438
Prepositional Phrase	15	186
Verb	332	8373
Other	19	71

Table 3.1: Statistics for the ERG lexical type hierarchy (leaf types only) and lexicon for the '1010' version of the ERG

correspondingly-named morphological phenomena,[3] while the punctuation rules track punctuation, which is treated by the grammar as word affixes. The '1010' version of the ERG has 197 construction rules and 68 lexical rules, with a more detailed breakdown shown in Table 3.2.

The ERG is described as a 'broad-coverage' grammar (by the standards of precision grammars), but obtaining reliable coverage figures is difficult for several reasons. Firstly, in common with many precision grammars, the coverage is strongly dependent on the domain of target text, as well as on the configuration of the preprocessing for POS-tagging, unknown-word handling, marking of named entities and dealing with non-sentential text. It is also dependent to a small extent on the memory and CPU speed of the machine on which the parser is run, since there is generally an upper-bound parsing time and memory limit. Additionally, the grammar is frequently updated, usually increasing the coverage, so the figures when using a grammar version

[3]Morphology, which was briefly mentioned in Section 2.2.4, is the study of word structure — in English, this usually means prefixes or suffixes attached to stem words. There is generally a distinction made between *inflectional* affixes, such as the plural suffix *-s* in *dogs*, and *derivational* (or *lexical*) affixes such as the adverb-forming suffix *-ly* in *magnificently*. Huddleston (1988:18) describes the main difference as being that inflectional affixes interact with syntax (as we saw with the agreement phenomena in Section 2.2.3), while derivational affixes essentially create new lexemes (and can thus change the broad word class, and have inflection affixes applied to them in turn). O'Grady *et al.* (1997:161) also notes that derivational affixes are generally less productive (applicable to fewer lexemes), but this is not particularly relevant, as only those affixes which happen to be quite productive are encoded as rules in the ERG.

Lexical Rules	68
Derivational	35
Inflectional	11
Punctuation	22
Construction Rules	197

Table 3.2: Statistics for the ERG syntactic construction rules and lexical rules for the '1010' ERG (leaf types only)

from one or two years previously can differ substantially. Baldwin *et al.* (2005) found that using a 2005 version of the grammar, coverage over closed-domain data such as that of Verbmobil project was around 80%. However over a random selection of 20,000 sentences of more complex text from the British National Corpus (Burnard 2000), the proportion of sentences for which there was full lexical coverage (without any unknown word handling) was 32%, and 57% of these sentences could be assigned a spanning parse by the grammar, giving coverage of 18%.

More recent work uses newer, more extensively developed versions of the grammar, and also adds unknown word handling along with some other preprocessing, both of which greatly increase the coverage. We have already noted coverage figures of 85% over Wikipedia (Flickinger *et al.* 2010). Additionally, in some of our work (MacKinlay *et al.* 2011b) based on the earlier '0902' grammar version, we reported coverage of 76% over biomedical text, while using the current '1010' grammar and updated preprocessing, we obtained coverage of 87% (MacKinlay *et al.* 2011a, the basis of Chapter 5). As well as this, when parsing the full-text of conference and journal papers in the NLP field, Schäfer *et al.* (2011) achieved coverage of 85%, presumably using a similar version of the grammar.

Finally, it is worth mentioning the licensing status of the ERG, which is advantageous for research purposes. It is open-source, as are a range of associated grammar development and parsing tools which we explain in more depth in Section 3.3, and can thus be freely used without requiring payment of any licensing fees.

3.2 Part-of-speech Taggers

Part-of-speech tagging, as discussed in Section 2.5.1 is important due to the unknown word-handling in our parsing setup, described in Section 3.3.6. Here we briefly outline two relevant taggers.

3.2.1 The TnT Tagger

TnT ("Trigrams 'n' Tags"; Brants 2000) is a POS-tagger based on a second-order *hidden Markov model* (HMM). A Markov model is a probabilistic model of states (which here are POS-tags) where the probability of being in a state depends only on the previous state while ignoring all earlier history. Since we have a second-order Markov model here, the previous state includes the previous two POS-tags. The fact that it is hidden means that these underlying states are not directly observable. However the associated observations (the words of the sentence) can be directly observed. TnT attempts to find the underlying tag sequence which maximises the probability of the combination of the observed words and corresponding hidden tag sequence. If we have a sentence of length T, with the words denoted $w_1, ...w_T$ and the tags denoted t_1, ... t_T (with t_0, t_{-1} and t_{T+1} as sentence boundary tags), then TnT calculates:

(3.1)
$$\arg\max_{t_1,...t_T} \prod_{i=1}^{T} \left[p(t_i|t_{i-1}, t_{i-2})p(w_i|t_i)\right] p(t_{T+1}|t_T)$$

The probabilities $p(t_i|t_{i-1}, t_{i-2})$ are estimated using frequency statistics based on unigrams, bigrams and trigrams (sequences of one, two and three tags) observed in the training data. For example if $f(X)$ denotes the frequency count of the sequence X, the observed probability of a tag given the previous two tags based on tag trigrams is given by:

(3.2)
$$p_{\text{obs}}(t_n|t_{n-1}, t_{n-2}) = \frac{f(t_{n-2}, t_{n-1}, t_n)}{f(t_{n-2}, t_{n-1})}$$

To account for unobserved tag sequences, the observed probabilities for unigrams, bigrams and trigrams are combined using weighted linear interpolation. Meanwhile, several word-internal features such as suffixes are used to improve the estimates of $p(w_i|t_i)$ for unknown words. The most likely tag sequence is then calculated using a beam-restricted variant of the Viterbi algorithm, a dynamic programming algorithm which creates a trellis of possible tag sequences and determines the most likely sequence. When trained on the PTB, TnT's token-wise accuracy of 96.7% achieves comparable performance to taggers based on more complex machine learning algorithms such as the maximum entropy approach of Ratnaparkhi (1996), although more recent work has slightly exceeded this (Toutanova *et al.* 2003; Giménez and Màrquez 2004).

3.2.2 The GENIA Tagger

The GENIA Tagger (Tsuruoka *et al.* 2005) is a POS-tagger optimised for robust POS-tagging across multiple domains, particularly targeted at biomedical text. Most principles of POS-tagging transfer straightforwardly across domains, so we could get respectable results by retraining a WSJ-tuned tagger on the same biomedical training data as used by the GENIA tagger, namely the GENIA corpus (described in Section 3.4.2 below) and the PennBioIE corpus (Kulick *et al.* 2004), another treebank of biomedical abstract. However the GENIA tagger is designed using lexical features (as we noted in Section 2.5.1, these are important for unknown-word handling) which are tuned to the biomedical domain. It is also able to robustly learn models based on training data from multiple domains, and still perform well over the test data from each of the training domains (Tsuruoka *et al.* 2005).

The tagging algorithm used by the current version of the tagger (Tsuruoka and Tsujii 2005) improves on that in the original work of Tsuruoka *et al.* (2005). This newer algorithm is another HMM-based approach like TnT, but is more sophisticated in that the HMM is generalised to be bidirectional and the "local classifiers", which are used to create the probability estimates in the multiplicand in the generalisation of equation (3.2), use a more highly-developed maximum entropy model.

Generalising the HMM to be bidirectional is achieved by allowing terms to depend on the preceding and following tags, with different possible combinations of preceding and following tags considered for a window-size of up to two tags away from a given token. This bidirectionality means that more information is potentially available to the local classifiers. The maximum entropy models in these classifiers have feature functions based on sequences of leading as well as following tags and these are used to estimate the probabilities within the bidirectional HMM. Classification using the full bidirectional HMM is relatively expensive since the different possibilities of preceding and following tags expand the complexity. Tsuruoka and Tsujii avoid this by applying a greedy optimisation strategy, first tagging the "easiest" (i.e. highest probability) tag conditioned on all known information, and repeatedly doing so until the entire sequence is tagged. This newer algorithm, trained on GENIA and WSJ data, achieves per-token accuracies of 96.9% over the WSJ corpus and 98.3% over GENIA.

The tagger has also been applied to Named Entity Recognition (NER) — the task of annotating salient entities such as company names or place names in free text. This is often useful for information extraction, but it can also be used in parsing, by treating a named entity as an atomic item, which means that internal structure within the named entity cannot interfere with the parsing process. In different domains, we are generally interested in different classes of NEs. In the biomedical domain, it is often protein and gene names which are of interest. The GENIA tagger as distributed has also been trained on the protein NE annotation from the JNLPBA shared task (Kim *et al.* 2004). This is not reported in Tsuruoka *et al.* (2005), so it is not clear whether this uses exactly the same features as the POS-tagging algorithm, but this

seems likely. It is possible to treat NER in a similar way to POS-tagging, by using *IOB-tags* (Ramshaw and Marcus 1995), where a token is tagged as 'O' if it outside an NE, 'B' if it begins one and 'I' if it is within an NE. The reported results[4] for the NER task are comparable to the best systems on the JNLPBA shared task.

3.3 Parsing and Treebanking Machinery

In this section we outline various open-source tools from the DELPH-IN collaboration that are used with the ERG within this thesis.

3.3.1 The PET Processing Platform

PET (Callmeier 2000) is a platform for HPSG-based language processing. It is primarily focussed on parsing, for which it provides a parsing component called CHEAP. It accepts input grammars defined in the TDL formalism, including the ERG. From a supplied input sentence, it determines a set of licensed HPSG analyses according to the grammar. The output can be stored as a *derivation tree* representing all of the edges in the chart which, along with the grammar, uniquely determines the complete HPSG AVM. It can also be stored as a representation of the semantics in MRS or some variant, which can be extracted from the HPSG analysis in a grammar such as the ERG.

CHEAP is a *bottom-up* parser, meaning that it assigns a structure to each token in the sentence and combines these into larger linguistic structures, successively combining items until a structure is found which matches a root node (i.e. starting symbol) as specified in the grammar.[5] CHEAP uses a *chart parsing* algorithm (Kay 1986). This means that it avoids repeatedly recreating sub-units of the parse trees as it explores the space of allowable trees by storing all nodes as they are created in a *chart*, and looking them up there as appropriate.

HPSG parsing, especially if not carefully implemented, is relatively expensive. This is primarily due to complexities of computation associated with applying unification operations to the large feature structures which arise out of broad-coverage grammars, as well as the memory management issues associated with storing and copying these feature structures. CHEAP has many design features to minimise the CPU time and memory usage for parsing. It is implemented in C++, to allow careful control of memory management, which the design takes advantage of by using an architecture designed to avoid dynamic memory allocation. It also uses the fastest unification algorithm from several alternative implementations tested (Callmeier 2000).

[4] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/tagger/

[5] This is in contrast to *top-down* parsers, which parse in the opposite direction, from the start symbol to the leaves; a bottom-up strategy is the only obvious approach for HPSG-based grammars which apply most constraints using the lexicon (Oepen and Carroll 2000)

In addition to this, PET as outlined originally contains implementations of various algorithmic optimisations to unification-based chart parsing proposed by various researchers. These include *quick-check*, a technique to find feature paths which most often cause unification failure (Kiefer *et al.* 1999) and *hyper-active parsing* (Oepen and Carroll 2000), an improved version of the parser-internal logic which attempts to avoid excessive numbers of unification operations and copying of feature structures.

PET is also under active development. Since the original 2000 publication, many further performance optimisations have been applied. One important set of such optimisations deals with the efficient handling of widespread ambiguity in parse forests. *Ambiguity packing* relies on the observation that a given structure from one tree may be shared by many other trees in the parse forest, even though none of them are globally identical. As applied to HPSG (Oepen and Carroll 2000), this means that when a feature structure found during parsing *subsumes* (i.e. is more general than) a subsequent feature structure, both feature structures can be *packed* into the same representation. This means that the parse forest can be created with a smaller memory footprint, and, crucially, with less copying, making parsing faster.

The complement to ambiguity packing is *selective unpacking* (Carroll and Oepen 2005; Zhang *et al.* 2007) which can be used when we have a parse selection model available to rank the parse trees. This model needs to be a particular class of maximum-entropy parse selection model whose feature functions are of a certain kind which is derivable from the parse forest (of the kind we examine in Section 3.3.5). Selective unpacking relies on the observation that it is possible to calculate this class of feature functions directly from the packed parse forest. It is then possible to only unpack those feature structures needed for the n-best trees in a way which is guaranteed to find the n globally optimal trees. Since we are often only interested in the top parse (i.e. $n = 1$) or occasionally the top ten, this can result in considerable savings in parse time as the CPU-intensive unpacking of the parse forest can mostly be avoided. This also enables the parsing of many highly ambiguous sentences which would otherwise fail due to exceeding memory thresholds or parsing time limits.

3.3.2 The Linguistic Knowledge Builder

The Linguistic Knowledge Builder, or LKB (Copestake and Flickinger 2000), is a grammar development platform. It includes a parsing module and reads grammars encoded in TDL, so has a substantial feature overlap with PET. While it includes many of the same optimisations to the parsing algorithm as in PET such as hyper-active parsing and ambiguity packing (Oepen and Carroll 2000), it is implemented in LISP and is not as highly-tuned for high-speed parsing performance. Instead, it is designed additionally for a broader range of tasks which are frequently undertaken during grammar development. It is targeted both at broad-coverage ERG-scale grammars (and is indeed the primary development platform for the ERG), as well as for smaller grammars as might be seen in a grammar engineering course.

It includes a GUI for visualising a wide range of linguistic objects useful in grammar development. It is possible to view AVMs associated with subtrees, lexical entries and grammar rules as well as parse trees for sentences and type hierarchies of the grammar. It also includes a number of features which are used within this thesis. Firstly, it has the ability (which is not present in PET) to *generate*[6] surface strings from a semantic representation, given a grammar. This is roughly the inverse operation to parsing to produce a semantic representation, which we cover in more detail in Section 3.5.5. It also has the ability to communicate with [incr tsdb()] (discussed in Section 3.3.3) about the loaded grammar, which is used in treebanking (Section 3.3.4).

3.3.3 The [incr tsdb()] Competence and Performance profiler

[incr tsdb()][7] (Oepen and Carroll 2000) is a 'competence and performance profiler' in the sense that it records the behaviour of a system, comprised of a grammar, test suites, and parser, at a particular point in time. 'Competence' is used here roughly in the sense of Chomsky (1957) — that is, the knowledge of the language held by a native speaker — but in this case referring to the grammar itself. [incr tsdb()] evaluates the grammar by applying it to test corpora and recording various statistics, including coverage, overgeneration and ambiguity. This enables grammar developers to find problematic aspects of the grammar, and to find areas which have progressed or regressed over time as the grammar is updated.

'Performance', on the other hand, refers to the system's overall efficiency for parsing. [incr tsdb()] communicates with the parser (which can be the LKB or PET, among others) and tracks a number of important metrics related to resource usage by the parser with the particular grammar, such as parsing time, memory usage and number of unification operations. These metrics and how they change are useful both for fine-tuning the parser software as well as for grammar engineers, who must consider efficiency issues as well as satisfactory linguistic explanations when designing broad-coverage grammars (Flickinger 2000).

The software also has more general uses than gathering statistics. It acts as a front-end to the parser including inputting sentences, invoking the parsing process, parallelising over multiple cores and storing the parser output in a specialised structured database. This means the linguistic information output by the parser is available alongside the gathered statistics. It is this ability to systematically store parse forests and relate them to input sentences which enables [incr tsdb()] to be used in the treebanking process, which we discuss in the next section.

[6] This is a slightly different sense from the term as applied to parsing using a context free grammar. Other synonymous terms include *tactical generation* and *realisation* (Carroll and Oepen 2005), although we generally use the shorter form *generation* when it is clear from context.

[7] Pronounced 'tee ess dee bee plus plus'

3.3.4 Constraint-based Treebanking

When using precision HPSG-based grammars such as the ERG, we generally make use of treebanks constructed by using "Redwoods-style" constraint-based treebanking (Oepen *et al.* 2004). This form of treebanking is quite different in nature to the methods used to construct treebanks such as the PTB, described in Section 2.4.1. It involves the treebanker selecting the best analysis from a possibly large number of alternative analyses for a sentence postulated by the corresponding grammar.

To treebank, the input is first parsed, and the (up to) 500 top-ranked parse trees according to some parse selection model are recorded. This set of parse trees is then presented to the human treebanker in the form of *discriminants* (Carter 1997; Oepen *et al.* 2002) determined by [incr tsdb()] from the parse forest. The discriminants used here correspond to instantiations of the lexical and syntactic rules of the grammar, as well as the lexical entries themselves, but only those that correspond to ambiguity in the parse forest and can thus discriminate between candidate parse trees.

During treebanking, the annotator confirms or rejects some subset of discriminants, and at each stage, [incr tsdb()] performs inference to automatically reject those discriminants that are incompatible with the current set of manually-selected and inferred discriminants. This means that each manual decision can directly or indirectly rule out a large number of trees, and the number of decisions required is on average proportional to the logarithm of the number of parses (Tanaka *et al.* 2005). The other advantage is each discriminant presented represents an opportunity for the annotator to make a decision, so the annotator can choose to make the easiest decisions first, possibly avoiding more difficult decisions (such as those relating to obscure lexical entries or syntactic rules) entirely, since the inference process will often rule them out before the annotator even needs to consider them.

The annotation interface, provided by the LKB, consists of two panes — one displaying a set of discriminants ordered by decreasing length (Zhang and Kordoni (2010) examine alternative ordering strategies), and one displaying the trees still remaining in the parse forest (if there are few enough to be tractably displayed). A screenshot of this interface is shown in Figure 3.1. In addition, once the parse forest has been reduced to a single tree, an MRS representation corresponding to the tree is shown. Rather than using feature structures or rule names from the grammar, these trees are displayed using node labels which look more like the conventional PTB-style labels ('NP', 'S', 'V' etc) and are derived from mappings in the grammar between feature descriptions and these labels (the ERG has 77 such mappings). These (relatively) compact and readable parse trees can be used by the annotators to confirm that the analysis is sensible once the parse forest has been sufficiently reduced; the MRS provides an alternative method of doing this. If there are no sensible analyses available in the grammar, the annotator also has the option of marking the sentence as having no grammatical parses — which can happen, for example, if a sentence uses a construction which is unknown by the grammar.

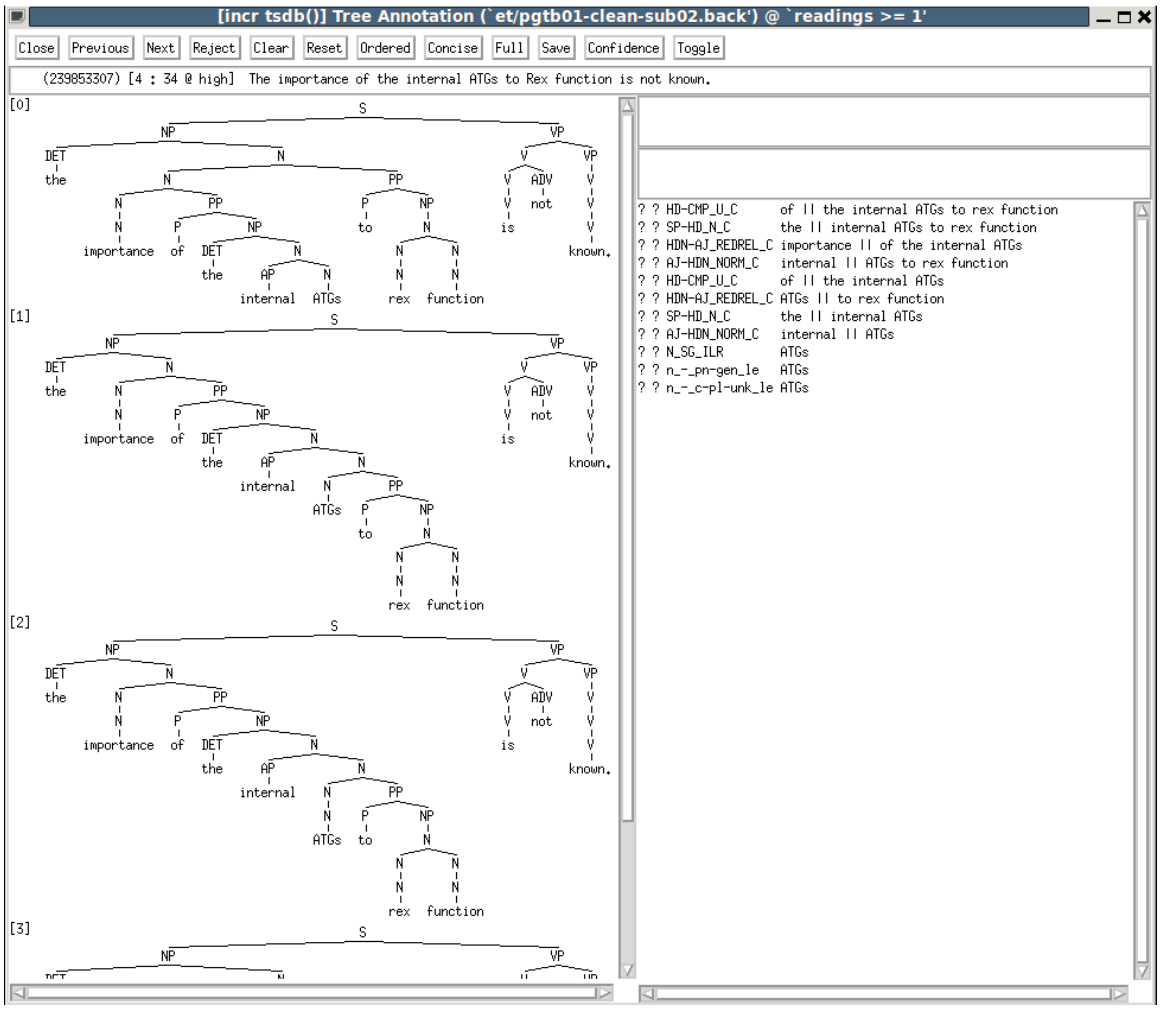


Figure 3.1: A screenshot of the LKB treebanking interface, where the parse forest has been reduced from 38 trees down to 4 trees, with the remaining trees displayed in the left pane, and the remaining discriminants in the right pane, which the annotator is able to select or reject.

At the end of the treebanking process we have a large number of rejected trees along with the single correct gold tree, for each sentence which was judged to have acceptable parses. This constitutes a Redwoods-style treebank, which has several advantages over a PTB-style treebank (Oepen *et al.* 2004). Firstly, it is is linked to a precision grammar implementing a particular theoretically grounded formalism. In particular, of course, this makes it useful for studying the formalism, but these rich annotations can also be used with other incompatible formalisms. We discussed some ways one could make use of superficially-incompatible linguistic annotation in Section 2.7 (although not in this particular way). A consequence of being linked to a precision grammar is that it is comparatively information-rich — there is a large amount of information embodied in the feature structure, derivation tree and semantic representation of the parsed sentence.

These treebanks can also be dynamically semi-automatically updated as the grammar changes. This is achieved by extracting constraints from the decisions made in the previous iteration of treebanking, and applying them to the parse forest created by the new grammar in order to disambiguate. This process generally removes most undesired candidate trees from the parse forest; some may be left if the new grammar has introduced extra ambiguities or incompatibilities which mean some of the previous round of decisions cannot be applied. The ability to dynamically update with minimal human intervention means that the treebank is not locked into some particular immutable set of syntactic conventions made at the time of the treebank creation, which then flow on to the treebank parsers which create them. The other way of looking at this, of course, is that it is *necessary* to update the treebank, in order to keep it completely compatible with the latest grammar version; this is a consequence of wanting a high-precision treebank linked to a precision formalism.

The treebank is also different in that alongside the correct trees, it includes negative training examples which have been affirmed by the annotators, in the form of the set of rejected trees. These correct and rejected trees can be used as positive and negative instances in the training data for building a discriminative parse selection model as discussed in the next section. This corresponds to using the data as a training corpus. Naturally, we can also use the set of disambiguated trees as a test or development corpus, by directly comparing trees with those manually annotated as correct.

3.3.5 Training and Applying Parse Selection Models

In parse selection using the methodology established by Velldal (2007) we use human-annotated incorrect and correct derivation trees to train a discriminative maximum entropy parse selection model, which discriminates between the set of all candidate parses to select the optimal parse tree. We feed all correct and incorrect parses (with the label for correctness preserved) licensed by the grammar to the TADM toolkit (Malouf 2002), and learn a discriminative maximum entropy model of the

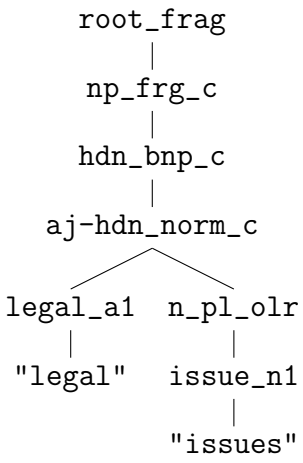


Figure 3.2: ERG derivation tree for the phrase *Legal issues*

kind described in Section 2.3.1 for selecting the best parse given a particular input sentence.

The features for training and applying the parse selection model are extracted from the candidate HPSG derivation trees, using the labels of each node (which are the rule names from the grammar) and those of a limited number of ancestor nodes. These features are created by examining each node in the tree along with its list of immediate children (which must be non-empty). The feature name is set to the concatenation of the node labels starting with the parent then the leftmost child. We also generally make use of *grandparenting* features, where we examine earlier ancestors in the derivation tree. A grandparenting level of one means we would also use the label of the target node's parent, a level of two means we would add in the grandparent label, and so on.

During the feature extraction, there is also an additional transformation applied to the tree. The immediate parent of each leaf, which is usually a lexeme, is replaced with the corresponding *lexical type*, a broader parent category from the type hierarchy of the grammar.

As an example of the feature extraction process, we examine the noun phrase *Legal issues* from the WeScience corpus, for which the correct ERG derivation tree is shown in Figure 3.2. For the node labelled `"issues"`, with grandparenting levels from zero to two, we would extract the features as shown in Figure 3.3 (where the parent node `issue_n1` has already been replaced with its lexical type `n_-c_le`, corresponding to a count noun).

Velldal (2007) notes the importance of significant thresholds related to the features. Grandparenting can improve performance and in at least some circumstances,

```
[(0) aj_-_i_le "legal"]
[(1) aj-hdn_norm_c aj_-_i_le "legal"]
[(2) hdn_bnp_c aj-hdn_norm_c aj_-_i_le "legal"]

[(0) n_-_c_le "issues"]
[(1) n_pl_olr n_-_c_le "issues"]
[(2) aj-hdn_norm_c n_pl_olr n_-_c_le "issues"]

[(0) n_pl_olr n_-_c_le]
[(1) aj-hdn_norm_c n_pl_olr n_-_c_le]
[(2) hdn_bnp_c aj-hdn_norm_c n_pl_olr n_-_c_le]

[(0) aj-hdn_norm_c aj_-_i_le n_pl_olr]
[(1) hdn_bnp_c aj-hdn_norm_c aj_-_i_le n_pl_olr]
[(2) np_frg_c hdn_bnp_c aj-hdn_norm_c aj_-_i_le n_pl_olr]

[(0) hdn_bnp_c aj-hdn_norm_c]
[(1) np_frg_c hdn_bnp_c aj-hdn_norm_c]
[(2) root_frag np_frg_c hdn_bnp_c aj-hdn_norm_c]

[(0) np_frg_c hdn_bnp_c]
[(1) root_frag np_frg_c hdn_bnp_c]

[(0) root_frag np_frg_c]
```

Figure 3.3: Features extracted at a maximum grandparenting level of two from the derivation tree in Figure 3.2, where lexical type mapping has occurred (mapping `legal_a1` to `aj_-_i_le` and `issue_n1` to `n_-_c_le`). Grandparenting levels are shown in parentheses, and the target node (for which we are extracting features) is emphasised for clarity.

a grandparenting level of 4 is optimal (Zhang *et al.* 2007) but it can exacerbate problems of data-sparseness, so the relevant threshold is generally determined empirically.[8] Another important feature-extraction parameter is the *relevance count threshold*. Following van Noord and Malouf (2004), a feature is relevant if it takes different values between known correct and incorrect parses, and thus is useful for discriminating between different outputs. The relevance count of a given feature is the number of input strings for which the feature is relevant, so by demanding a minimum value for this over the whole corpus, we have control over which features are included in the model. We return to this parameter selection problem in Section 4.4.1,

Once the model is trained, the parse ranking process is conceptually fairly simple. We extract the same features for the parse trees produced for a given input sentence. Recall the general probability calculation shown in (2.31). In this instance, we can treat a particular derivation tree t as the instance label, and calculate the score in this instance as shown in (3.3):

(3.3)
$$p_\lambda(t|s) = \frac{1}{Z_\lambda(s)} \exp\left(\sum_i \lambda_i f_i(t)\right)$$

where s is the input sentence and λ_i is the i'th feature weight as before. Note that f_i now only depends on t, as the feature functions all take their values from substructures within the parse tree, as we outlined earlier in this section. Also note that in parse selection, we are generally not concerned with the actual value of $p_\lambda(t|s)$. This means that we do not need to calculate the value of the normalisation constant $Z_\lambda(s)$, since this is constant for a given sentence s. It is not even necessary to take the exponent of $\sum_i \lambda_i f_i(t)$ — we can simply use the value of the sum to obtain a score for the sentence. Since the exponential function is monotonically increasing, this will preserve the relative rank, so the score can then be used to order the derivation trees according to the maxent model.

In most cases, however, we do not exhaustively score every tree in the parse forest. Many long, real-world sentences have enough ambiguity that they are difficult to parse exhaustively in PET without running into resource constraints. It is also generally unnecessary, as most applications would get relatively little value out of having thousands of parse trees. Instead, the scoring of the trees in this fashion occurs in conjunction with the selective n-best unpacking in PET outlined in Section 3.3.1, but this procedure, by design, is guaranteed to find the globally optimal trees in ranked order according to this scoring algorithm. Most downstream applications use only the best-ranked tree, or occasionally the top ten. The primary exception is when we are parsing for the purposes of treebanking, when, as mentioned earlier

[8] Toutanova *et al.* (2005) note that ERG trees are relatively deep, due in part to the fact that rules are all unary or binary, so features based on information from higher levels are important.

in this section, we use the top-500 trees, so that there is a better chance that the correct tree is within the subset of the parse forest available to the treebanker, and to minimise bias from the parse selection model of the previous iteration.

3.3.6 Preprocessing and Unknown Word Handling

Preprocessing of some kind is important for most parsers. They usually expect that the input is split into sentences in advance, and that these sentences in in turn are split into tokens, usually following the tokenisation conventions of the PTB. Parsers can also have a POS-tagging phase before parsing (Collins 1996; Briscoe *et al.* 2006) (although not always — Rush *et al.* (2010) presents a model for simultaneous POS-tagging and parsing) and allow named entities to be explicitly annotated (Briscoe *et al.* 2006).

These techniques are all available (or required) for parsing with the ERG. It is expected that the input is split into sentences in advance, but apart from this, the ERG is able to accept raw textual input and produce a most-likely interpretation (in conjunction with a parser and a parse selection model). However, in this configuration, the parsing will fail if any unknown words[9] are encountered in the input as the grammar has no means of handling these. It is not feasible to expect grammar engineers to manually add all possible words of a language to the lexicon, and even when using automated techniques to extract lexical entries from other resources, there will inevitably be gaps.

POS-tagging is a commonly used method for handling this. The text is pre-tagged with some default POS-tagger — often TnT from Section 3.2.1 — which outputs PTB-style POS-tags, with possibly multiple tags per token. PET, as well as accepting plain text, allows a range of input formats which allow the POS to be specified. The ERG includes a set of mappings between these POS-tags and *generic lexical entries*, and instantiates one of these when it sees a POS-tagged unknown word. The generic lexical entries are designed to act as very flexible and ambiguous instances of the general lexical types corresponding to the part-of-speech. So, for example, verb-like tags such as 'VB' and 'VBN' are mapped to generic entries for transitive verbs with optional complements, meaning they are agnostic about whether complements appear or not. This means, of course, that the entries are less constraining than those manually specified in the grammar. This is unavoidable since the information is not available, but is also not a particularly severe problem unless there is a large number of unknown words in a sentence, as the other known items in the sentence will apply more constraints to the allowed parses.

Another source of possible parse failure is named entities, possibly containing whitespace or other non-alphabetical characters which interfere with the parsing pro-

[9]Here we use 'unknown words' to mean unknown in the lexicon of the grammar, rather than unknown in the training corpus, which is less of a problem for a handcrafted grammar.

cess. These entities may have internal structure but it can not generally be sensibly analysed by a general-purpose natural language parser. Certain generic lightweight named entities are handled internally within the grammar using *chart-mapping rules* (Adolphs *et al.* 2008), which allow the grammar to specify mappings from regular expressions to feature structures which can then be parsed as normal. For example, the ERG maps time expressions such as *11:23am* using this method to an appropriate generic lexical entry, avoiding the possibility of the parser incorrectly decomposing the expression, and also ensuring that sensible semantics can be assigned. There are other classes of named entities which are not so easily handled by simple regular-expression matching. These are often domain-specific such as names of chemical compounds, and best handled by a domain adapted NER system. PET has input formats which allow such named entities, possibly spanning whitespace, to be explicitly marked as such, so they can then be treated as atomic generic unknown words by the grammar.

3.4 Treebanks

3.4.1 ERG Treebanks

There is a range of corpora available for the ERG in various domains and sizes. Here we describe those that are relevant to this thesis.

The LOGON corpus

There the LOGON corpus (Oepen *et al.* 2004) is a collection of English translations of Norwegian hiking texts from the LOGON project (Lønning *et al.* 2004). It is freely available for download, and contains 8535 sentences which have exactly one gold standard tree annotated in the treebank.[10] The ERG was extensively developed for the LOGON project (particularly in terms of lexicon) so the grammar may show a slight bias towards this particular corpus as it was implicitly tuned for it, and as such, we would expect the corpus to be easier for the ERG to parse.

The WeScience Corpus

The more recent WeScience corpus (Ytrestøl *et al.* 2009) is a set of Wikipedia articles related to computational linguistics, which is, again, freely downloadable. With 9126 sentences with a single gold-standard tree, it is slightly larger than the LOGON corpus in the number of sentences, and has somewhat longer average sentence length. It is in a very different domain, and beyond both corpora exhibiting fairly formal written prose, there is little similarity in content (we examine this in more detail in Section 3.4.1).

[10] This omits six small sections 'jhu', 'jhk', 'psu', 'psk', 'tgu' and 'tgk'.

Corpus	Description	Example
WeScience (WeSc)	Wikipedia articles	*There are a number of competitions and prizes to promote research in artificial intelligence.*
LOGON (LOG)	Hiking brochures	*The bird life is typical of an area of this kind.*
Cathedral and Bazaar (C&B)	Essay on Linux development	*I had been preaching the Unix gospel of small tools, rapid prototyping and evolutionary programming for years.*
Robot1 (robot1)	Transcribed interactive dialogue	*okay I walked through the hallway um turned right*

Table 3.3: Corpora we use for our experiments and example sentences from each.

Other Corpora

There are also smaller corpora we use. The Cathedral and Bazaar corpus,[11] an essay on open-source software development. This treebanked corpus is also included with the ERG distribution. Finally, there is one more corpus which is not yet publicly available, labelled 'robot1', which consists of transcripts of several spoken dialogues between two humans, one of whom is simulating a robot collaborating in a virtual world task of hunting for coloured blocks (Flickinger *et al.* 2009). As natural dialogue, the utterances in this corpus are on average relatively short and full of disfluencies, and many are not full sentences. This makes it quite different to the carefully edited written prose that comprises the other three corpora we use. The various corpora are described in Table 3.3.

Corpus Characteristics

In Table 3.4 we give broad statistics such as counts of tokens (using the tokenisation in the gold-standard ERG output). We also list the ambiguity of the sentences in terms of the number of parses postulated by the ERG (counting at most 500 parses per sentence, and hence underestimating), giving an indication of how difficult the parse selection task is beyond the rough estimates we can make on the basis of sentence length. robot1 again stands out as being particularly different in this regard, with fewer than 100 per sentence, while LOGON is intermediate in difficulty compared to the progressively more ambiguous sentences of WeScience and C&B, in line with the increased sentence lengths in these corpora.

[11] http://www.catb.org/esr/writings/cathedral-bazaar (authored by Eric Raymond).

	WeScience	LOGON	C&B	robot1
Total Sentences	11558	9410	770	1537
Parseable Sentences	10220	8799	767	1412
Validated Sentences	9126	8535	567	1303
Parses per valid sent	274.1	236.5	322.8	100.7
Tokens per sentence	15.0	13.6	18.7	5.8

Table 3.4: Corpora used in this thesis showing total number of sentences, how many of those can be parsed, and of those how many are 'validated', with a single gold parse. Also shown is average sentence length (in tokens) and average ambiguity on test set (number of parses produced per sentence with each sentence capped at 500 parses – this approximates, but underestimates, the difficulty of the parse selection problem)

3.4.2 The GENIA Treebank

The GENIA treebank (GTB: Tateisi *et al.* (2005)) was created in recognition of the fact that many supervised learning techniques suffer a performance drop when the domain of the training data is mismatched with test corpus — and most domains are mismatched with the newswire domain of the PTB WSJ corpus which is used for many NLP tasks. The problem is especially noticeable on domains with unique characteristics, such as biomedical text. An annotated treebank in the appropriate domain could help to address these issues.

The GTB is something like an analogue of the PTB in the domain of biomedical abstracts. The sentences annotated are the complete text from the GENIA corpus (Kim *et al.* 2003), a collection of 1,999 abstracts from MEDLINE returned by a search for the keywords *human*, *blood cells* and *transcription factors*. The abstracts have been sentence-split, and manually annotated with PTB-style labelled constituent structures, but are encoded in XML rather than the parenthesised text format of the PTB.

The annotation guidelines are derived from the PTB II guidelines (Bies *et al.* 1995), with some simplifications. Certain "functional tags" indicating semantics of adverbial phrases are not used. Additionally, there are simplifications to the noun phrase structure — the PTB node labels 'NX' and 'NAC' which relate to NP structure are not used, and in general the internals of NPs are left as unstructured as possible, with the main exception being instances of co-ordination, where the NP structure affects the remainder of the sentence. One of the reasons for this decision was that the annotation was performed by annotators without biomedical expertise. According to Tateisi *et al.*, with sufficiently strict annotation guidelines, most decisions can be

resolved linguistically, but the internal structure of NPs stands out as being particularly difficult to resolve. Additionally it would be possible to subsequently have an annotator with domain knowledge annotate the NPs with internal structure. Indeed, Vadas and Curran (2007) found that it was possible to perform post hoc addition of NP structure to the Penn Treebank, so there is no particular reason why such an approach should not be applicable to the GTB as well.

3.5 Minimal Recursion Semantics

3.5.1 MRS

Minimal Recursion Semantics, or MRS (Copestake *et al.* 2005), is a semantic framework that was designed originally for integration with deep grammars, particularly those that utilise typed feature structures such as HPSG. By their nature, deep parsers are capable of producing a large amount of information, which we would ideally like to preserve for later processing. At the same time, the ambiguity inherent in natural language means that there will invariably be aspects of the semantics which cannot be resolved completely from the source text.

One priority in the design of MRS was for use in symbolic machine translation systems such as for the Verbmobil project (Wahlster 2000). This means that we wish to able to construct *transfer rules* to convert between MRSs for different language, and that it should facilitate tactical generation (which is also a design goal in its own right). We thus wish to avoid, as much as possible, one sentence with multiple semantic representations, or making the semantic representation so general that it allows invalid logical forms. MRS aims to allow an appropriate level of specificity in the meaning representation corresponding to a particular sentence. It allows highly specific representations where these are valid, but also enables representations of scope ambiguity as well as allowing other forms of underspecification (arising, for example, from an unfilled argument slot). At the same time it is designed to remain computationally tractable in such cases of ambiguity.

An MRS representation of a sentence or smaller unit of language (abbreviated, like the formalism, as MRS) is described as a flat structure (contrasted in Copestake *et al.* (2005) with tree-like structures such as those of first-order predicate calculus) which includes of a bag of *elementary predications* or EPs. Each EP consists of a label, a relation name, and a set of arguments. The MRS also includes *handle constraints*, specifying constraints on the relationships of the arguments to each other. This essentially means that it can specify a partial tree-like ordering on the elements while at the same time allowing underspecification of this ordering, thus offering a compromise between entirely hierarchical and entirely flat representations and their respective shortcomings. Each MRS can also be converted into usually multiple formulae in a propositional logic-style representation. We do not give a complete

treatment here — the reader is referred to Copestake *et al.* (2005) for a more detailed account. Here we give an overview, focussing on practical aspects which are relevant to this thesis.

Figure 3.4 shows an example MRS obtained for the sentence *Kim gave every pony a sweet carrot*. Looking first at the top-level elements, we can see that most of the semantic information is encoded in the RELS attribute, which contains the list of EPs. This is augmented by the handle constraints listed under the HCONS attribute. The two other top levels attributes were not mentioned in the earlier discussion. LTOP ("local top") is primarily used during compositional creation of the MRS, and refers to the topmost label which is not the label of a quantifier.[12] There is also an INDEX attribute introduced by the grammar, which points to a privileged non-handle label, and is also used for the mechanics of compositional semantics (as discussed in Section 3.5.4).

In an EP such as the one introduced by ``_give_v_1_rel''<4:8> in the figure, we can see all of the components listed above. ``_give_v_1_rel'' is the relation, <4:8> are character indices[13] to the source sentence, h8 is the label, and the arguments are e2, x6, x10 and x9. Equality of labels indicates that the corresponding EPs form a conjunctive relationship and are always treated together.

There is always an ARG0 which is afforded special status, referring to the *characteristic variable* (Copestake 2009) introduced by the EP. Each variable which appears in a given MRS is the characteristic variable for exactly one EP. For verbs and adjectives this is an event variable, and for nouns this is the entity to which the noun corresponds. Subsequent arguments are labelled according to the relation of the argument to the predicate. Arguments can be variables such as e19 or x10 (where the first letter indicates the nature of the variable — e referring to events and x to entities), or *handles* such as h17. Note that handles cannot be characteristic variables, and thus will not occupy the ARG0 slot. We can also see from Figure 3.4 that where the variables are introduced, some semantic properties similar to those we might see in an AVM are also shown, indicating salient semantic attributes such as number and tense as appropriate.

The arguments of each EP are deliberately labelled in a semantically non-committal fashion in the ERG, although it would equally be possible to assign more meaningful roles (however, it is difficult to come up with a canonical list, and doing so would burden the grammar engineer with maintaining consistency). There are some standard conventions for these neutral arguments — for verbs, ARG1 is the *deep subject* of the verb, often corresponding to the syntactic subject. Here, the ARG1 of the verb is x6, which we can see is the same variable as introduced

[12] A similar GTOP feature for "global top" is also part of the MRS formalism, but we do not cover it here as it is redundant in the ERG (Copestake *et al.* 2005:22).

[13] Character spans are not mentioned in Copestake *et al.* (2005) but are provided by the current version of the grammar and are useful for many downstream applications. They are not particularly relevant to the theory of MRS.

```
[ LTOP: h1
INDEX: e2 [ e SF: PROP TENSE: PAST MOOD: INDICATIVE PROG: - PERF: - ]
RELS: <
       [ proper_q_rel<0:3>
         LBL: h3
         ARG0: x6 [ x PERS: 3 NUM: SG IND: + ]
         RSTR: h5
         BODY: h4 ]
       [ named_rel<0:3>
         LBL: h7
         ARG0: x6
         CARG: "Kim" ]
       [ "_give_v_1_rel"<4:8>
         LBL: h8
         ARG0: e2
         ARG1: x6
         ARG2: x10 [ x PERS: 3 NUM: SG IND: + ]
         ARG3: x9 [ x PERS: 3 NUM: SG IND: + ] ]
       [ _every_q_rel<9:14>
         LBL: h11
         ARG0: x9
         RSTR: h13
         BODY: h12 ]
       [ "_pony_n_1_rel"<15:19>
         LBL: h14
         ARG0: x9 ]
       [ _a_q_rel<20:21>
         LBL: h15
         ARG0: x10
         RSTR: h17
         BODY: h16 ]
       [ "_sweet_a_to_rel"<22:27>
         LBL: h18
         ARG0: e19 [ e SF: PROP TENSE: UNTENSED MOOD: INDICATIVE ]
         ARG1: x10
         ARG2: i20 ]
       [ "_carrot_n_1_rel"<28:34>
         LBL: h18
         ARG0: x10 ] >
HCONS: < h5 qeq h7 h13 qeq h14 h17 qeq h18 > ]
```

Figure 3.4: MRS for the sentence *Kim gave every pony a sweet carrot*

by `named_rel<0:3>`, corresponding to the proper noun *Kim*, which is indeed the syntactic subject. The relation `‘‘_give_v_1_rel’’` corresponds to a verb which takes two syntactic complements, and these align with semantic arguments, so there are two additional arguments `ARG2` and `ARG3`, corresponding to the item given and the recipient respectively. We can see that `ARG2` is the item given by matching the variable `x10` with the EP for which it is `ARG0: ‘‘_carrot_n_1_rel’’<28:34>`; we could do the same for `ARG3`. A few particular kinds of relations have different sets of arguments. Quantifiers have `RSTR` and `BODY` arguments, corresponding to the *restriction* and *body* of the quantifier, particular aspects of the logic of quantifiers (which we return to later in this section). Conjunctions such as *and*, *or* and *but not* have `L-INDEX` and `R-INDEX` arguments for the left and right conjoined items attached to the conjunction, and point to variables corresponding to these conjoined items.

The relation names are determined by the grammar, but there are certain regular conventions associated with them respected by the ERG. The names prefixed with ‘_’ are for relations which directly correspond to lexemes used in the sentence, while those without the underscore prefix are *grammatical predicates* automatically inserted by the grammar for semantic coherence when there is no directly-matching lexeme, such as named entities, noun compounds and some quantifiers. Relations for open-class words (nouns, verbs, adjectives and adverbs) are divided into three underscore-separated components before the final `_rel`: the bare name, the part-of-speech, and the sense[14] of the lexeme (represented numerically or by a mnemonic string). Closed class words and relations for grammatical predicates omit the sense label.

The handles mentioned above are used in the *qeq constraints*, which relate a handle to a label, indicating a particular kind of outscoping relationship between the handle and the label — either that the handle and label are equal or that the handle is equal to the label except that one or more quantifiers occur between the two (the name is derived from ‘equality modulo quantifiers’). These scoping constraints are used for the `RSTR` of quantifiers. Using handles directly in these circumstances is not sufficiently general for relatively complex reasons related to theoretical semantics (Copestake *et al.* 2005:10). The decision on the exact nature of these constraints was also made to facilitate building up semantics using typed feature structures, using the method we outline in Section 3.5.4.

These qeq constraints indicate the outscoping relationships between the EPs, without needing to specify more detail than is implied by the sentence. For example, `_every_q_rel` has a `RSTR` argument of `h13`. By combining this with the qeq constraint `h13 qeq h14` and the fact that the `LBL` of `_pony_n_1_rel` is `h14`, we know that `_every_q_rel` outscopes `_pony_n_1_rel`. Using these constraints, we can derive various *scoped MRSs* (six in this case), corresponding to the different interpretations

[14]This is only used where there are senses distinguishable by syntax (Copestake 2009), not the narrower senses often used in word-sense disambiguation tasks.

of quantifier scope. We show two of these interpretations in (3.4) and (3.5), where '∧' denotes conjunction.

(3.4) `_a_q(x10, _sweet_a_to(e19, x10, i20) ∧ _carrot_n_1(x10), _every_q(x9, _pony_n_1(x9), proper_q(x5, named(x5, Kim), _give_v_1(e2, x5, x10, x9))))`

(3.5) `_every_q(x9, _pony_n_1(x9), _a_q(x10, _sweet_a_to(e19, x10, i20) ∧ _carrot_n_1(x10), proper_q(x5, named(x5, Kim), _give_v_1(e2, x5, x10, x9))))`

These scoped MRSs look more similar to traditional predicate logic formulae. In this particular example, there is scope ambiguity concerning whether there was a single carrot which was given to each pony (when the determiner `_a_q` has the wider scope) as shown in (3.4), or whether all ponies were given an individual carrot (wide scope `_every_q`) as shown in (3.5), but these and the other four undisplayed interpretations are succinctly captured in the original MRS.

3.5.2 Robust MRS

Closely related to MRS is Robust Minimal Recursion Semantics, or RMRS (Copestake 2004), which is designed to be reversibly convertible to MRS (provided a valid MRS can be formed), while at the same time allowing the more ambiguous representations that are desirable in certain contexts. The primary difference is that in RMRS the arguments of the EPs mentioned above are split up, separating the arguments from the predications themselves. Base predicates are unary, instead of having an arity dependent on the individual predicate, and arguments are indicated by binary relations to these predicates. The utility of this is that RMRSs are more easily produced by shallower processing techniques and these have certain advantages over the deeper techniques that are the primary focus here.

An example RMRS is shown in Figure 3.5. The correspondence to the matching MRS from Figure 3.4 should be reasonably clear. The EPs are introduced by the lower case strings followed by parentheses, such as '`_pony_n_1(h14,h10005,x9:3:SG:+:)`'. The EP name is similar to the name from the MRS, although omits the '`_rel`' suffix for lexical predicates. The first item '`h4`' within the parentheses is the *label*, analogous to the `LBL` attribute of MRSs, with equality between labels having the same meaning. The second item, '`h10005`' is the *anchor*, which functions as a unique ID which can be used to refer to each predicate. The labels are not always unique since label equality is meaningful,[15] as we see with `_sweet_a_to` and `_carrot_n_1` in the example, so are not suitable for this purpose. These unique IDs are needed since the arguments

[15] Alternatively, it is also permitted to indicate label equality by using *in-group* constraints, rather than actually having the labels as equal, but we do not show this here.

```
h1
proper_q_rel(h3,h10001,x6:3:SG:+:)
  RSTR(h10001,h5:)
  BODY(h10001,h4:)
  qeq(h5:,h7)
named_rel(h7,h10002,x6:3:SG:+:)
  CARG(h10002,Kim)
_give_v_1(h8,h10003,e2:PROP:PAST:INDICATIVE:-:-:)
  ARG1(h10003,x6:3:SG:+:)
  ARG2(h10003,x10:3:SG:+:)
  ARG3(h10003,x9:3:SG:+:)
_every_q(h11,h10004,x9:3:SG:+:)
  RSTR(h10004,h13:)
  BODY(h10004,h12:)
  qeq(h13:,h14)
_pony_n_1(h14,h10005,x9:3:SG:+:)
_a_q(h15,h10006,x10:3:SG:+:)
  RSTR(h10006,h17:)
  BODY(h10006,h16:)
  qeq(h17:,h18)
_sweet_a_to(h18,h10007,e19:PROP:UNTENSED:INDICATIVE:)
_carrot_n_1(h18,h10008,x10:3:SG:+:)
  ARG1(h10007,x10:3:SG:+:)
  ARG2(h10007,u20:)
```

Figure 3.5: An example RMRS corresponding to the MRS in Figure 3.4 for the sentence *Kim gave every pony a sweet carrot*

are specified separately to the EPs for ease of constructing using shallow techniques. The third item, '`x9:3:SG:+:`' in this case, is the variable introduced by the EP, with a similar meaning to the MRS, along with attribute-value pairs for the properties of the variable.

The arguments in the RMRS are indicated by capitalised strings using the argument names familiar from the MRS such as `ARG1(h10007,x10:3:SG:+:)` in the example. The first item '`h10007`' within the parentheses is a reference to the anchor of an EP, in this case `_carrot_n_1`. These arguments can be specified in any order, but have been placed below the corresponding EP and indented for readability. The second item '`x10:3:SG:+:`' refers to the variable which actually fills this argument slot for the EP. The qeq constraints operate identically to those in the MRS.

Copestake (2004) gives some examples of deriving RMRSs from shallower representations. RASP (Briscoe *et al.* 2006) is a parser designed for robustness which is generally described as being an intermediate level parser on the shallow–deep continuum, since it does not attempt to handle complex language phenomena. There is a method outlined for deriving RMRSs from the parsed output of RASP, based on applying rules keyed off the roughly 400 phrase structure rules in RASP. A simpler method is also described for deriving (very underspecified) RMRSs from a POS-tagged sequence of words, by constructing predicates on the basis of the orthography of the word and its part of speech, and populating the arguments while making as few assumptions as possible. Frank (2004) presents a slightly more complex and flexible approach to the same problem which may be advantageous in certain situations, although neither paper presents any empirical evaluation.

3.5.3 Dependency MRS

Dependency MRS, or DMRS (Copestake 2009), is a graph-based MRS variant based on dependencies. Like other dependency representations, it uses labelled links between nodes in a graph to represent the relationship between elements of a sentence. These graph links replace the variables used in (R)MRS to indicate arguments and quantifiers of EPs and how they are related. It is still closely tied to MRS — it is designed so that an MRS representation can be freely converted in a DMRS representation containing minimal redundancy (via RMRS), and vice versa. This lossless transformation requires annotating the links with additional information apart from simple information about arguments. Specifically, annotations derived from label equality and from the qeq constraints are also included.

The conversion is enacted by creating three graphs between the EPs representing label equality, qeq constraints and linkages via variables. To create the variable graph, a non-quantifier EP, which we will call the "stand-in EP", is used to stand in for its characteristic variable (i.e. its `ARG0`). For a given EP, a non-characteristic variable v used in its arguments is converted into a directed link, pointing towards the stand-in EP for v (i.e. the EP which has v as its `ARG0`). In other words, the variables are

removed by making each non-quantifier EP replace its own `ARG0` variable, and linking other arguments to this stand-in EP. The link is labelled with the argument name from the original EP. Any variables which are not used in more than one EP are ignored — this may arise, for example, if an argument slot of some lexeme is not filled, such as the variable `i20` in Figure 3.4. This means that the conversion from MRS to DMRS is lossy, since we are no longer sure of the arity of the predicate, but this information is recoverable from the grammar by matching up the relation name when it required for the reverse conversion.

This variable graph is merged with the other two graphs, and the links from these can mostly be converted into suffixes on the labels of the links from the variable graph. These suffixes are `EQ` for label-equality, `NEQ` for label inequality, and `H` for a handle constraint (which is always a qeq constraint here). For further details of the procedure, the reader is referred to Copestake (2009). The result of applying the procedure to the MRS in Figure 3.4 is shown in Figure 3.6.

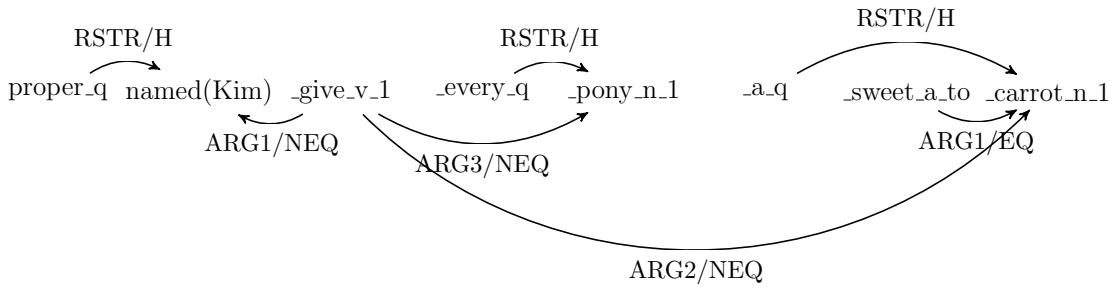


Figure 3.6: Example DMRS for sentence *Kim gave every pony a sweet carrot*, converted from the MRS in Figure 3.4. Node names have been abbreviated, while character spans and properties have been omitted.

DMRSs have a number of advantages compared to the other MRS variants. They are more readable at a glance, with similar readability to other dependency representations. The format may also be more appropriate for many information extraction tasks as it can require less inference to determine the required information. In applications where end users need to specify relations, a DMRS-like representation is probably the most appropriate of the MRS family; indeed, it was used for this reason by Schäfer *et al.* (2011).

3.5.4 Building Semantics using Feature Structures

In Section 2.2.4, we outlined HPSG, a syntactic formalism, and in Section 3.1.1, we discussed the ERG, an implementation of it. In Section 3.5, we also described a semantic formalism, MRS, which is designed to be used with constraint-based grammars. However, we have not yet discussed the link between these — i.e. how it is possible to derive MRS semantics using a constraint-based grammar.

To explain how the compositional construction of semantics works, we need two tools on top of what we described about MRS in Section 3.5. Firstly, we need to know that MRSs can be represented in feature structures. Since this is one of the design criteria, this is unsurprisingly a straightforward task. The MRS shown in Figure 3.4 was derived from a feature structure, where each of the attributes such as RELS and HCONS was a feature, with values set to nested feature structures or lists of feature structures, and equality of variables indicated by co-indexation tags. An example of the conversion is shown later in this section. In the grammar itself, the MRS for each lexical item is specified by the grammar engineer as a feature structure within the lexicon.

We also need a way to combine the semantics of these individual items into a representation for the semantics of an entire sentence. For this, there is an algebra of MRS composition, described formally in some detail by Copestake *et al.* (2001). Using this, the semantics of each individual feature structure can be created in conjunction with the parsing operation. The rules of this compositional algebra are integrated into the rules of the grammar so that the MRS of each constituent is created as the rules of the grammar are applied.[16]

This is of course a different way to use feature structures and unification than for the purely syntactic analysis we showed in Section 2.2.4. In syntactic analysis, the information in the features is in principle available in the output, but it is used primarily to apply constraints to the allowed parse trees. We do not pay attention to many of the feature values in the resultant feature structure for the sentence (indeed, many features are "consumed" by unification operations lower down in the parse tree). However, in semantic composition we are using these features to recursively build a semantic representation, and the value of these semantic features is explicitly something which we do care about, as it is often the reason we are parsing the sentences in the first place.

The formal details of the description of the compositional creation of semantics in Copestake *et al.* (2001), but we give an idea of the basic principles, which are also outlined in Copestake *et al.* (2005). Each lexical item has a distinguished main EP, known as the *key* — this is enforced by the grammar. The LTOP handle of the MRS is generally equal to the label of the key, except in the case of *floating* EPs,[17] which generally correspond to quantifiers, when the LTOP is a separate non-equated handle.

To create the MRS for a phrase from the components, there are two cases. In the common case of *intersective* combination, which applies to modification and verb

[16] Another method (Bos *et al.* 2004) for extracting interpretable (first-order logic) semantics from parse trees is to apply post hoc analysis to the derivation structures output by the parser, which is less intrinsic to the parsing process, in contrast to this tightly-integrated syntax-semantics interface, for which MRS is explicitly designed (Copestake *et al.* 2005) (The standard dependency structures produced by CCG are somewhat more closely integrated with the grammar than the first-order logic forms).

[17] So-called because of their freedom in the MRS with respect to the qeq constraints.

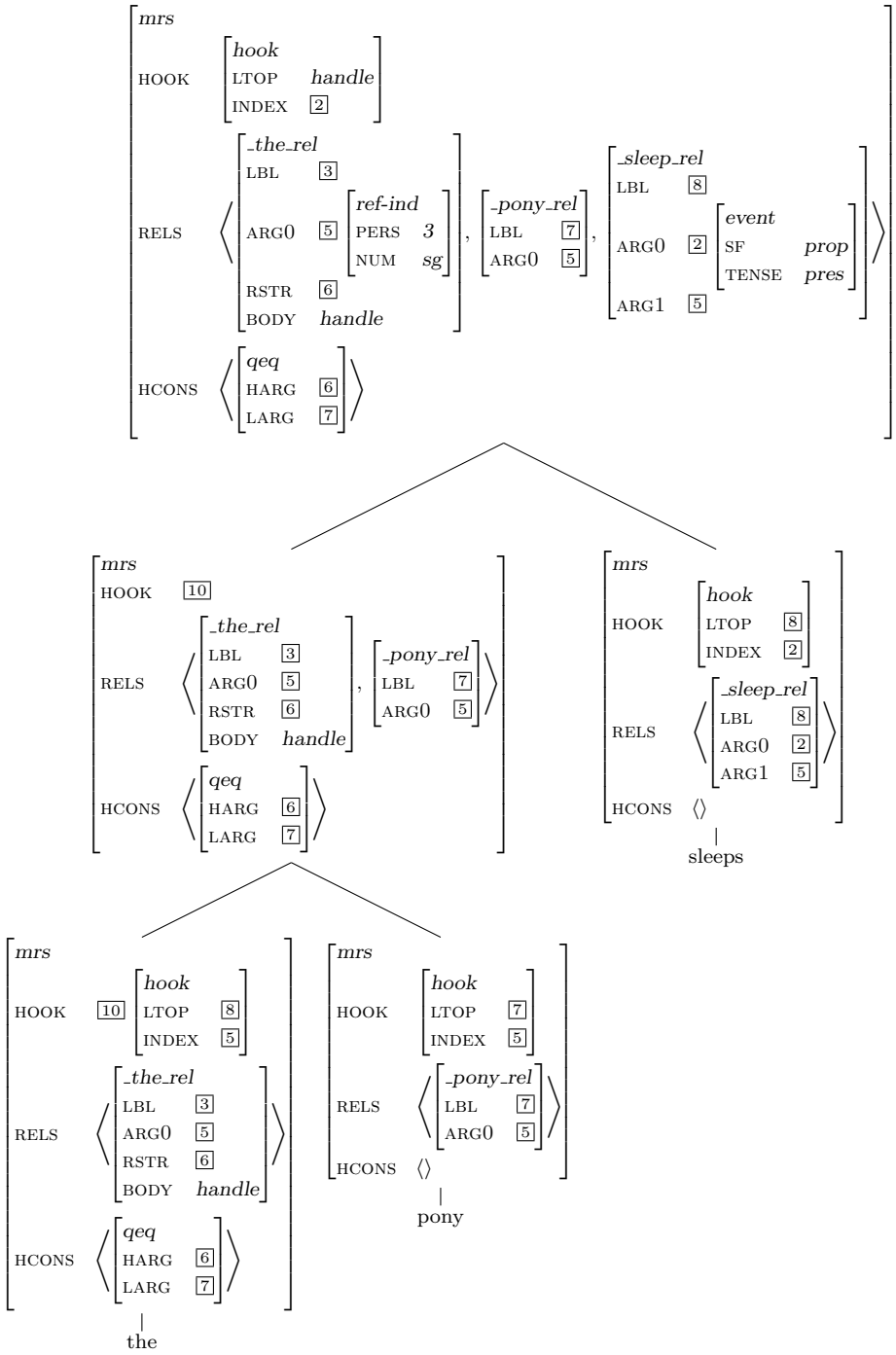


Figure 3.7: Demonstration of semantic composition using feature structures to create an MRS for *The pony sleeps*. Unary rules which do not modify the MRS have been omitted, and the feature structures labelled 2 and 5 have been truncated.

```
[ LTOP: h1
  INDEX: e2 [ e SF: PROP TENSE: PRES MOOD: INDICATIVE PROG: - PERF: - ]
  RELS: <
         [ _the_q_rel
           LBL: h3
           ARG0: x5 [ x PERS: 3 NUM: SG IND: + ]
           RSTR: h6
           BODY: h4 ]
         [ "_pony_n_1_rel"
           LBL: h7
           ARG0: x5 ]
         [ "_sleep_v_1_rel"
           LBL: h8
           ARG0: e2
           ARG1: x5 ] >
  HCONS: < h6 qeq h7 > ]
```

Figure 3.8: MRS for the sentence *The pony sleeps* corresponding to the top-level feature structure in Figure 3.7. For readability, the AVM tags indices match the variable indices (e.g. 'x5' corresponds to 5).

complements, RELS and HCONS are treated in the same way — in each case the value in the mother is set to the appended values of all of the daughters (implemented in HPSG through unification). The LTOP values of the daughters are equated with each other and with the value of the mother (also implemented in HPSG grammars through unification, using the shared-structure tags such as '1' which we saw in Section 2.2.4).

The other case is *scopal combination* which is used where one of the daughter EPs scopes over another (the argument daughter), as quantifiers and certain adverbs such as *probably* do. The rules for scopal combination are slightly more complex. The lists of RELS and HCONS are appended as in intersective modification. However, LTOP of the parent is equated only with the scoping daughter, and another qeq constraint is added to the HCONS list stipulating that the handle argument of the scoping EP outscopes the label of the argument EP. Further detail on both of these is available in Copestake *et al.* (2005:13). The grammar is also permitted to introduce additional semantic content as the rules are applied, which is how, for example, conjunctions are handled (Copestake *et al.* 2005:28). In Figure 3.7, we show an example of compositional construction of complete sentence MRS from components MRSs using feature structures. We show the top-level MRS feature structure converted into a more familiar MRS representation in Figure 3.8. Note how the indexes of the variables and

AVM tags match (e.g. 'x5' corresponds to [5]) — this is done for readability, but does not need to be the case, as these indexes are arbitrary.

The mechanisms to create MRSs during parsing are integrated into the ERG and other DELPH-IN grammars. Primarily, MRS has been used with HPSG, although there is no reason in principle why it cannot be used with other constraint-based formalisms — indeed, it has been applied to NorGram, an LFG analysis of Norwegian (Lønning *et al.* 2004).

3.5.5 Generating Sentences from Semantics

We noted in Section 3.3.2 that the LKB is capable of *generation*, which is in some ways the inverse operation to parsing. Rather than creating a semantic representation from a string of text, we create a string of text from a semantic representation. The processes are not perfect inverses — in both directions we generally get ambiguity. We discussed methods of handling this ambiguity for parsing in Section 3.3.5, by using a statistical model to pick the best parse. In generation, there are two primary methods of handling ambiguity. One is avoiding spurious ambiguity, which we briefly describe later in this section, and is important for efficiency as well as reducing the space of candidate output sentences to choose from. For unavoidable ambiguity, it is also possible to use similar methods to those of Section 3.3.5 (augmented with some extra statistical information) to rank the generated candidates against a statistical model. This is explored in detail for the candidate sentences generated using the ERG by Velldal (2007), but as it is not directly relevant for this thesis, we do not discuss this further here.

The generation component of the LKB is discussed extensively by Carroll *et al.* (1999). The LKB uses a *lexically-driven generation* algorithm, which takes an MRS input[18] and in the lexical lookup stage, maps each EP to a lexical entry in the grammar. This lexical entry is instantiated and inserted into a *generation chart* (somewhat analogous to a parsing chart) which is then populated with the compatible syntactic structures. Lexically driven generation has a number of advantages compared to other candidate approaches. It is appropriate for lexicalised grammars such as the ERG, and the generation algorithm can be reasonably generic to support multiple different grammars fairly easily. On the other hand this approach can have efficiency problems if not carefully implemented. We discuss later in this section how the LKB is designed to avoid these.

The lexical lookup stage is, according to Carroll *et al.*, the semantic analogue of morphological processing in parsing. The generator indexes lexical entries by the relations they contain. The relations in the input semantics are checked against this index, and any matching lexical entries are retrieved, with the variables in the relations instantiated corresponding to those in the input MRS. There is also an

[18] Carroll *et al.* note that other formalisms could in principle be used with their algorithm

index of lexical rules and grammar rules which introduce semantic content, and after instantiating the entry, any such appropriate rules are applied if they match the input semantics. Ideally the correspondence between MRSs and input semantics is one-to-one, but there are many reasons why this might not be the case. One is that relations might be introduced by grammar rules instead of lexical entries — for example, the ERG inserts a dummy quantifier for bare noun phrases. In the LKB, such rules are instantiated if they match the input semantics. Secondly, some lexical items do not introduce relations — for the ERG, the auxiliary verb *has* is an example of this in sentences such as *The pony has slept*, where the auxiliary is marked as a property on the event variable corresponding to the main verb *slept*. Similar to the inverse case, these relation-less lexical entries are only added if licensed by a rule.

These lexical entries are instantiated into a generation chart. After lexical instantiation, this chart is partially populated, and a bottom-up chart generation algorithm is invoked. The difference between a generation chart and a parsing chart is that a generation chart is constrained by the semantics of the input MRS rather than orthography of the input sentence. The algorithm for populating the chart, then, is roughly analogous to what happens in parsing: grammar rules which allow existing edges to combine are instantiated. The primary difference is that here we have additional constraints from the semantics (instead of positions in the input string). These are determined on the basis of variables in the input MRS; if a grammar rule is incompatible with these constraints, it is not considered. This procedure is applied repeatedly, as in parsing, until all possible analyses which match the root condition have been found. The surface form for each compatible analysis can then be created from the orthographic representation stored in the grammar.

We have omitted discussion of some important optimisations to this process. Intersective semantics, which we see, for example, when adjectives modify nouns, can create an exponential (or worse) explosion in the number of candidate realisations. This occurs when there is more than one intersective modifier for a word, as in the *The small brown pony*. The corresponding MRS has separate EPs for *small* and *brown* modifying *pony*, so useless edges such as one corresponding to *The small pony* will be generated in various combinations. Carroll *et al.* detail how the chart generation process is altered so that this form of modification is delayed until after the other parts of the chart have been populated. There are also many subsequent optimisations present in the LKB discussed by Carroll and Oepen (2005). These include augmenting the input MRS with transformations to ensure that no time is wasted generating analyses incompatible with the output semantics, and efficiently representing ambiguous structures using packing (as discussed in Section 3.3.1).

3.6 Evaluation Metrics

NLP research on parsing is often treated much like a supervised learning task, concerned with reproducing the labelling on the test set. Thus, research into more accurate parsing is concerned with finding ways to choose the parse of a sentence which more closely matches the gold standard in some test corpus. On average over the whole corpus, we wish to improve the closeness of these matches. However, there are many ways we can define and measure the closeness of the match; in this section we review some of these.

3.6.1 Top-N Exact Match

Previous work using the ERG and related DELPH-IN grammars has generally reported results on the basis of exact match of the top parse or exact match within the top-10 parses (Zhang *et al.* 2007), in contrast to other measures we discuss in the following sections which measure partial matches. There are a number of factors contributing to this usage of exact match. In particular, it is useful for reflecting the utility of a ranking model for treebanking. In an ideal treebanking process, it is important to select the exactly correct tree. A granular metric which awards partial credit to a mostly correct tree (such as those we discuss in Sections 3.6.2 and 3.6.3) could be misleading in this case, since a tree which has most constituents or dependencies correct is nonetheless still not the tree which the treebanker should select, and it is important to know whether the exact tree is present in the parse forest.

In Redwoods-style treebanking, as discussed in Section 3.3.4, the parse selection model is crucial for two reasons. Firstly, a correct parse close to or at the top of the ranking enables the treebanker to quickly select it as the gold-standard parse tree. Secondly, the treebanking process requires the selection of some ad hoc cutoff for the number of highest-ranked parses to present to the treebanker, which is usually set to 500. This number decides the balance between tractability in the treebanking process for the treebanker (and, to a lesser extent, the underlying machinery), and completeness in terms of not excluding correct but low-ranked parses. Inevitably, there will be a small amount of 'leakage' — correct parses which are not ranked high enough to be considered — but we can reduce this while holding the number of parses constant by providing a better parse selection model. So, better parse selection models enable treebanks to be built more quickly and with greater coverage.

Throughout this thesis, we use notation Acc_N to denote the exact gold-standard tree being found somewhere in the top-N parses. In terms of treebanking utility, Acc_{500} tells us whether the correct analysis is available to the treebanker in the top-500 analyses; conversely, it is possible to calculate the average N required to achieve, say, 80% exact match parsing accuracy. However, these values are expensive to calculate for dozens of different parser configurations. Following Zhang *et al.* (2007), in this

thesis, we generally show scores for the less-expensive Acc_1 (primarily) or Acc_{10}, which we suggest can be used as proxies for Acc_{500} (a question we return to in Section 4.4.2). Both of the figures also tell us other important information about the treebanking utility of the parse selection model: whether the target parse is ranked highest, or occurs within a highly-ranked manageable group of trees.

3.6.2 Constituent Scores

Not all evaluation depends on a notion of matching trees exactly. Much evaluation enables the assigning of partial credit, depending how close to the gold-standard tree a given parsed tree is. One way to achieve this is based on producing similar constituents in the phrase structure tree. Since many treebanks, parsers and grammars are based on some underlying notion of constituency structure, it makes sense in evaluation to use some notion of matching constituents between the trees output by the parser and those in the gold-standard treebank.

The simplest case makes the most sense when there are no systematic differences between the parsed trees and the gold trees, as should be the case with treebank parsers. It is also the basis of what has become known as PARSEVAL (Black *et al.* 1991) despite the the term not begin used in that paper, although we return to the differences later in this section. We simply convert each test and gold tree into a set of constituents. Those constituents which are identical in terms of endpoints (according to token span, or possible character span) between the test and the gold tree are correct. We can then calculate precision and recall in the usual way:

(3.6)

$$\text{precision} = \frac{\#\text{ correct constituents}}{\#\text{ constituents in test tree}}$$
$$\text{recall} = \frac{\#\text{ correct constituents}}{\#\text{ constituents in gold tree}}$$

If we ignored the node label in determining constituent correctness, these figures are for *unlabelled precision* and *unlabelled recall*. We can also require that the labels of the constituents match in determining correctness, giving the *labelled* variants of these. In each case, we could calculate the *F-score* as the harmonic mean of these values. A related metric is *crossing brackets*, which is the number of constituents in the test parse which violate constituent boundaries in the gold parse — in this case, lower scores mean a closer tree match.

The aforementioned PARSEVAL metric is designed specifically for evaluating analyses for different parsers against a common test corpus, even when they make different assumptions about handling particular phenomena. Unlike the generic procedure outlined above, it is optimised especially for English, in that it includes a set of

language-specific simplifications designed to abstract away the possibly different assumptions made by grammar engineers. Essentially, it calculates unlabelled precision and recall as described above, but includes many simplifications, such as removing brackets from single-token constituents as well as removing punctuation tokens, the negation particle *not*, pre-infinitival *to*, and the possessive clitic *'s*.

Collins (1996) uses what is described as PARSEVAL, citing Black *et al.* (1991), although the use of labelled precision and recall conflicts with the stipulation of Black *et al.*. In addition, while punctuation is excluded, it seems to be not the strict version of the metric described above. Much subsequent work (e.g. Charniak 1997; Clark and Hockenmaier 2002; Charniak and Johnson 2005) ostensibly using PARSEVAL seems to follow this scheme, using either labelled or unlabelled precision and recall and ignoring punctuation but not applying the other transformations, although most of this later work does not explicitly refer to the Black *et al.* paper. The term 'PARSEVAL' has thus become (at least since Collins (1996)) a conventional shorthand for comparing labelled or unlabelled precision and recall across constituents while in general ignoring punctuation, often using the `evalb` evaluation tool[19], rather than a strict adherence to Black *et al.* (1991). In any case, as we noted earlier in this section, the details of the remapping are particularly important when there are different assumptions made by the parser and the treebank, which is generally not the case for treebank parsers, so these deviations are not likely to have a large effect.

3.6.3 Dependency-based Metrics

Constituent-based metrics similar to PARSEVAL have become a de facto standard amongst treebank parsers. However, there is another important metric allowing partial credit for incorrect trees, based on *dependencies*. These are linkages based on the syntactic or semantic structure in sentences. They correspond roughly to (usually labelled) directed edges in some graph-based representation of the sentence, which is generally derived directly from the parse tree (thus having an indirect correspondence with the semantics), although it is also possible for this to come from the induced sentence semantics explicitly. These edges are represented as tuples, and the standard precision, recall and F-score measures are applied to these tuples, replacing the constituents in Equation (3.6). Dependency representations are sometimes used in conjunction with constituent-based evaluation, for reasons of appropriateness to a particular formalism, or as an alternate view of the effectiveness of the parser (Collins 1999; Clark and Hockenmaier 2002).

Collins (1999: Section 7.8), recognising that constituent-based evaluation does not accurately measure the accuracy of attachment phenomena, presents an alternative evaluation using syntactic dependencies extracted from the parse tree.[20] These de-

[19] `http://nlp.cs.nyu.edu/evalb`

[20] These are related to but distinct from those used in the parse selection model.

pendencies are triples denoting the relationship between a phrasal head (which has been heuristically identified) and a "modifier".[21] The triples are composed of the index (within the sentence) of the head word, the index of the modifier word and the *relation*. The relation element in turn has four subelements — the parent, head and modifier non-terminal labels (with all POS-tags mapped to 'TAG'), and the direction of the dependency. An advantage of this tuple-based evaluation is that it enables constructing patterns over the relation elements to see how well the grammar is discerning particular relations in aggregate. These dependencies are by their nature very local, with each dependency only reflecting the relationship between two levels of the tree.

Clark and Hockenmaier (2002) investigate evaluation for a CCG-based parser using CCGBank (Hockenmaier and Steedman 2002), a version of the PTB adapted for CCG. They present results for PARSEVAL-style evaluation against CCGbank, but note that this is not particularly meaningful for CCG for two reasons. Firstly, CCG allows multiple different tree structures for equivalent analyses, which would nonetheless be penalised by a constituent-based metric. Additionally CCG trees (like those of HPSG) are unary or binary branching, meaning the structure is deeper and contains more brackets than the corresponding PTB tree, increasing the chance of crossing brackets.

They present two dependency metrics which avoid these problems. The first is a close CCG analog of the approach of Collins outlined above. The second uses the "deep" dependencies output by the CCG parser of Clark *et al.* (2002). These include many long-distance as well as local dependencies, for example in mapping particular argument slots of verbs to directly to the head noun which is the complement of that verb. These mappings are created by annotations in the grammar which link syntactic arguments to dependency relations. As such, they could be considered more semantic in nature (although Clark *et al.* (2002) do not make claims about this). Each dependency created by the grammar and used in the evaluation is a 4-tuple, consisting of the head word (not necessarily a linguistic head), the *functor category*, the argument slot, and then head of the argument. The functor category is a CCG type somewhat analogous to a lexical type in the ERG. It encodes information about the arguments expected by the item, and in the grammar of Clark *et al.* (2002), an annotation of the head as well as the corresponding numerically indexed dependency relations. The argument slot is a numeric index corresponding to one of the dependency relations. For example a transitive verb would have two dependency relations, in its functor category corresponding to the subject ('1') and the object ('2'). Both types of dependencies can be straightforwardly converted to unlabelled dependencies by ignoring the tuple elements which do not correspond directly to words in the sentences (and can thus be thought of as edge labels).

[21] This is a broader meaning of *modifier* than we have previously been using, referring to an element related to a head, including what we defined in Section 2.2.4 as a *complement*

The evaluation metric of Miyao and Tsujii (2008) using the induced HPSG Enju (discussed in Section 2.4.2) is fairly similar to the deep dependencies of Clark and Hockenmaier (2002). It evaluates over tuples denoting predicate-argument structure extracted from the feature structure of the parse sentence. Each tuple contains the head word of the predicate, the head word of the argument, the type of the predicate (such as "transitive verb") and the argument label (e.g. 'MODARG' or 'ARG2') All of these have close analogs to the elements of the deep CCG dependency tuples, although as Miyao and Tsujii note, since they are encoding different relations and different levels of granularity, they are still not directly comparable.

Elementary Dependency Match

Elementary Dependency Match (EDM: Dridan and Oepen 2011) is a dependency evaluation method which is explicitly based on semantics of the sentence, although there is some similarity to the partially syntactic dependency metrics such as those described above.[22] It is closely tied to the MRS formalism outlined in Section 3.5.1, and as such, is applicable, to some extent, to the parsed output of any grammar which produces MRS representations, including the ERG which is the focus of this thesis.

EDM divides semantic information into three classes, which are each named mnemonically as shown:

- **class 1** (ARGS): the core semantic predicate-argument structure — this is the most similar to the class of information evaluated by most dependency evaluation schemes, although they are generally more syntactic in nature.
- **class 2** (NAMES): predicate properties, such as lemma and word class.
- **class 3** (PROPS): properties of entities within the sentence.

EDM is evaluated over triples, each of which corresponds to one of these classes. These are derived from *elementary dependencies* (EDs) (Oepen and Lønning 2006), a variable-free variant of MRS. These are somewhat similar to links in a DMRS graph, as the conversion to EDs from an MRS is based on characteristic variable associated with a predicate. However, equality and qeq constraints are ignored, meaning that the conversion is not reversible (and nor is it intended to be).

EDM uses character spans rather than token indexes to identify sentential elements, allowing for different tokenisations in competing analyses. Each EDM triple links one of these spans to some other entity via some link name. Each ARG triple comes from a link between a span and another span. The triple consists of the first span, the argument name (or "role") such as 'ARG1' and the second span. NAME triples link a span (the first element) to the corresponding predicate name from the

[22] In fact, Bos *et al.* (2004) argue that the deep dependency representations of Clark and Hockenmaier (2002) are a reasonable proxy for semantic accuracy

MRS (the third element), with the second element of the triple mapped to 'NAME'. Finally, each PROP triple consists of a span, a property name (such as 'PERSON' or 'TENSE)', and a property value. These triples are derived from elementary dependencies primarily by replacing each (predicate, span) pair in the ED with just the character span, however this information is preserved in the NAME triples which link from each character span to each predicate. This means that the task of making sure arguments match is decoupled from the task of correctly identifying predicates. An example set of EDM triples is shown in Table 3.5.

In the evaluation, we simply have a set of triples extracted from the MRS of the gold tree and a set from the test tree, and we can calculate precision and recall in the usual way. It is also possible to weight the different classes differently — for example if we want to prioritise the matching of the ARG class.

A common configuration is to weight ARGS and NAMES equally, and ignore PROPS. This is designed to be as similar as possible to other deep dependency metrics, such as Clark and Hockenmaier (2002) and Miyao and Tsujii (2008), in that they include broadly the same classes of information. However, there are differences in the details of how they are evaluated. The 4-tuples of Clark and Hockenmaier and Miyao and Tsujii are most similar to the EDM ARG tuples, but they include some kind of syntactic category as an element of the tuple. A mismatched syntactic category would therefore cause a test tuple to not match the gold tuple. The closest analog to this category in the ERG or other DELPH-IN grammars would be the lexical type. This is not included in the ARG tuples, but a close approximation to this is implicitly encoded in the NAME tuples, as the relation name can be mapped back to the lexical entry which in turn uniquely determines the lexical type. However, EDM differs in that this category information is encoded in separate tuples, and applies to nominal elements as well.

3.6.4 Comparison of Evaluation Metrics

As noted in Section 3.6.1, exact-match metrics have been the de facto standard for evaluation over DELPH-IN grammars for many reasons including treebanking utility. However the exact match metric may not be ideal for all evaluation — it is very unforgiving, as it gives zero-credit for a tree which may only be incorrect in some minor aspect.

The constituency and dependency metrics we described in contrast have a notion of partial correctness of a tree. This is probably desirable when we using the parser output in downstream applications, as even a partially correct tree almost certainly encodes some useable information. How closely such a metric corresponds with performance on a downstream task is, of course, an empirical question, which we return to in Chapter 6.

We have not described a constituent-based approach for the ERG. It is straightforward to create a PARSEVAL metric (in the loose sense), but since the semantics is

Class	Start	Role	End
ARGS	⟨0:3⟩ (*Kim*)	ARG0	⟨0:3⟩ (*Kim*)
	⟨4:8⟩ (*gave*)	ARG1	⟨0:3⟩ (*Kim*)
	⟨4:8⟩ (*gave*)	ARG2	⟨28:34⟩ (*carrot*)
	⟨4:8⟩ (*gave*)	ARG3	⟨15:19⟩ (*pony*)
	⟨9:14⟩ (*every*)	ARG0	⟨15:19⟩ (*pony*)
	⟨15:19⟩ (*pony*)	ARG0	⟨15:19⟩ (*pony*)
	⟨20:21⟩ (*a*)	ARG0	⟨28:34⟩ (*carrot*)
	⟨22:27⟩ (*sweet*)	ARG1	⟨28:34⟩ (*carrot*)
	⟨28:34⟩ (*carrot*)	ARG0	⟨28:34⟩ (*carrot*)
NAMES	⟨0:3⟩ (*Kim*)	NAME	proper_q
	⟨4:8⟩ (*gave*)	NAME	_give_v_1
	⟨9:14⟩ (*every*)	NAME	_every_q
	⟨15:19⟩ (*pony*)	NAME	_pony_n_1
	⟨20:21⟩ (*a*)	NAME	_a_q
	⟨22:27⟩ (*sweet*)	NAME	_sweet_a_to
	⟨28:34⟩ (*carrot*)	NAME	_carrot_n_1
PROPS	⟨0:3⟩ (*Kim*)	PRES	3
	⟨0:3⟩ (*Kim*)	NUM	sg
	⟨0:3⟩ (*Kim*)	IND	+
	⟨4:8⟩ (*gave*)	MOOD	indicative
	⟨4:8⟩ (*gave*)	PERF	–
	⟨4:8⟩ (*gave*)	PROG	–
	⟨4:8⟩ (*gave*)	SF	prop
	⟨4:8⟩ (*gave*)	TENSE	past
	⟨15:19⟩ (*pony*)	PERS	3
	⟨15:19⟩ (*pony*)	NUM	sg
	⟨15:19⟩ (*pony*)	IND	+
	⟨22:27⟩ (*sweet*)	MOOD	indicative
	⟨22:27⟩ (*sweet*)	SF	prop
	⟨22:27⟩ (*sweet*)	TENSE	untensed
	⟨28:34⟩ (*carrot*)	PERS	3
	⟨28:34⟩ (*carrot*)	NUM	sg
	⟨28:34⟩ (*carrot*)	IND	+

Table 3.5: Example EDM triples from the MRS in Figure 3.4. Textual strings are shown solely for readability — only character spans are used in evaluation

also available, and is the recommended way to use the parser outputs, an evaluation which directly reflects the semantics seems more appropriate, so EDM is an important evaluation metric throughout this thesis. On the other hand, we have noted above why exact-match is also useful in some circumstances, so we also report exact match results; this additionally facilitates comparison with previous work. In Section 4.3.2, we compare the two families of metrics empirically.

3.7 Summary

In this section, we have provided a reasonably in-depth review of the grammar, software, corpora, semantic formalisms and evaluation metrics which are most relevant to this thesis.

The grammar we covered in Section 3.1.1 was the ERG, the handcrafted HPSG-based grammar of English which is used in all subsequent chapters of this thesis. We use it with an associated family of software packages, which we covered in Sections 3.2 and 3.3. As described in Section 3.4, it also has several treebanks of various sizes available. The output from the grammar which we most commonly use is in a semantic formalism known as MRS, or a freely-convertible variant, which are all described in Section 3.5. Finally in Section 3.6 we covered the various evaluation metrics available for evaluating the quality of the outputs of the grammar.

Part II

Parse Selection

Introduction

In many tasks within natural language processing, the start-of-the-art approaches are based firmly within the supervised learning paradigm of machine learning. Broadly, as we explained in Section 2.3, this means that the task is framed as a kind of classification problem. A learning algorithm is supplied with some set of pre-labelled training data, and from this data it learns a model of how to apply these labels automatically to unseen instances.

Supervised learning approaches produce state-of-the-art performance (allowing for tradeoffs between speed and incremental accuracy improvements) in a diverse range of tasks in NLP, including POS-tagging (Brants 2000; Toutanova *et al.* 2003; Tsuruoka *et al.* 2005) and syntactic parsing (Charniak and Johnson 2005; Clark and Curran 2004), which is our primary concern here.

The impressive performance of supervised learning approaches makes them extremely attractive to researchers. However, by definition, these require a pre-existing, manually-annotated dataset. For parsing, this generally takes the form of a treebank of parse trees where the structure has been manually assigned by humans (although various labour-saving shortcuts can be used, such as post-correcting automatically-created output). Creating these resources requires expensive annotation labour from experts, and for such resources to be useful for learning to correctly assign parse trees, they must be relatively large, which makes creating a dataset an expensive proposition.

Nonetheless, given the importance of such resources for NLP, there have been a large number of independent efforts to create such treebanks (annotated with either phrase structure trees or dependencies) for a range of languages and domains. One of the best-known and most widely used is the Penn Treebank (Marcus *et al.* 1993) of newswire English, but there are many others English-language treebanks, including the GENIA treebank (Tateisi *et al.* 2005) and the Redwoods treebank (Oepen *et al.* 2004) discussed in Section 3.4.1. There is also a large range of treebanks of other languages, including the Alpino Dependency Treebank of Dutch (van der Beek *et al.* 2002), the Hinoki Treebank of Japanese (Bond *et al.* 2004), the TIGER treebank of German (Brants *et al.* 2004) and the Penn Chinese Treebank (Xue *et al.* 2005), among many others. The development of such treebanks has likely been encouraged by the success of the Penn Treebank and other precursors in enabling high-quality parsing and POS-tagging, and recognition of the value of creating such resources in new domains and languages for achieving similar goals.

These datasets, which collectively embody countless hours of labour from the creators, are highly valuable resources, and in general they are highly reusable, which is by design. Their existence has enabled a huge amount of subsequent research which depends heavily on annotated corpora, including, for example, the parsing and POS-tagging algorithms mentioned above. The primary use of these is to enable the train-

ing of machine-learning algorithms, as previously mentioned, but they also provide a source of testing/development data to evaluate the quality of the improvements.

In many cases, the corpus coverage seems sufficient that creation of new treebanks is of fairly marginal benefit for learning a parsing model. However, this is only true in a particular set of circumstances: primarily, the language of the treebank must match the target application. Secondarily, if there are strong differences in domain between the source treebank and target application, the value of the corpus is considerably reduced. A third dimension of possible difference is the formalism of the treebank, which may superficially seem to make the treebank of little value. The first kind of mismatch is difficult to work around — if there is no treebank in target language, a new treebank is required.[23] However the second two issues are more surmountable. In Section 2.6, we reviewed methods of domain adaptation, and in Section 2.7 we noted some ways in which we can still extract useful data from treebanks which use formalisms that are incompatible with our target application.

This part of the thesis is concerned with exploring the interactions of mismatched domains and mismatched formalisms in the context of deep parsing which is the primary focus of this thesis. In Chapter 4, we explore how out-of-domain training data affects performance in deep parsing, and ways to avoid the problem. In Chapter 5, we examine ways to use data from incompatible formalisms to improve parse selection and reduce the labour requirements for treebanking in deep parsing.

[23]However, Nakov and Ng (2009) found that resources from closely-related languages could be useful for statistical machine translation involving resource-poor languages, so we might expect a similar technique to be useful for treebanks. In addition, Xia and Lewis (2007) found it was possible to semi-automatically create dependency structures for resource-poor languages, by using interlinear glosses. They parse the English translation using an off-the-shelf parser and align the source sentence to the English translation using hints from the interlinear gloss. The English parse is then projected using the word alignment to create syntactic structures for the target language.

Chapter 4

Domain Adaptation for Parse Selection

4.1 Introduction

In the introduction to Part II we noted that supervised learning techniques are very important in modern NLP, particularly for state-of-the-art syntactic parsing. The resources needed for this are valuable, both in terms of being particularly expensive to produce but also particularly useful once they are created. We also noted that the domain of a treebank is one way in which a particular treebank can be suboptimal for some target application. Understandably, there is a desire to make maximal use of data where it exists, but this is not always straightforward. If this training data does not match the target domain it still has value, but it is not as useful as in-domain data. For an illustration of why this might be the case, consider:

(4.1) *A natural rock wall divides the path into the cave.*

(4.2) *A heuristic algorithm divides the path into separate components.*

(4.1) is the kind of sentence which could come from the LOGON corpus, while (4.2) is created to look like a sentence from WeScience. Note in particular that the POS sequences in the sentences are similar, and there is substantial overlap of lexical items. However, the prepositional phrase headed by *into* attaches at a different point in the sentence in each case. In (4.1), it modifies *path*, while in (4.2) it modifies *divides*, but the lexical entries at both possible points of attachment are identical in each sentence. If we had many occurrences of *path into* ***X*** in the training corpus, the parser would be more likely to choose the correct parse for (4.1) (all else being equal, which is seldom the case in NLP) and the incorrect parse for (4.2). This is a situation where the domain of the training corpus could be important. The domain of hiking brochures such as LOGON may be more likely to refer to a *path into* ***X*** than a

corpus of encyclopaedic articles on computational linguistics such as WeScience, in which case LOGON could be a more appropriate training domain for parsing (4.1) and WeScience could be more appropriate for (4.2). This is of course an artificial example, and actual parse selection models handle the interactions of many thousands of parameters rather than the one parameter we are referring to here, but this should serve to illustrate why out-of-domain data can be suboptimal for parsing a particular target domain.

In such a situation, various strategies can be adopted to apply the out-of-domain data to the target domain, since we still wish to make use of as much training data as possible. It is possible to simply ignore the performance penalty and accept the out-of-domain data as close enough to the target data, and this is indeed the strategy much work has adopted in the past. However since Gildea (2001) pointed out the extent of this performance penalty, there has been a large amount of research on domain adaptation of treebanks, which we reviewed in Section 2.6.

Much of this previous work has looked at parsing with treebank-derived grammars, and it is not clear how applicable the results are to parsing with hand-crafted precision grammars. In this chapter, we investigate this question, exploring domain adaptation in the context of parse selection for HPSG-based precision grammars, focusing in particular on the English Resource Grammar, or ERG, described in Section 3.1.1.

We share with much other work the fairly common concern with maximising parser accuracy in order to improve performance in downstream applications which make use of the parser output — grammar consumers wish to know how much accuracy they can expect when moving across domains. However, we also have an extra reason to be concerned about parse selection accuracy. In Redwoods-style constraint-based treebanking (discussed in Section 3.3.4), the parse selection model is important for the treebanking process. In particular we require the target tree to be in the top-N trees which we examine or it is not possible to select a tree for the sentence. If we can reduce the value of N while still being fairly confident of seeing the best tree within the top-N, we can reduce the time required to create the treebank. Alternatively, we may already have N set to the highest-feasible value in terms of computability of the parse forest and tractable treebanking time; in this situation we wish to maximise the possibility of the best tree being within the top-N. Either way, having the best-possible parse selection model is valuable for treebanking, which means that if we have ways to adapt to new domains, we can treebank for those domains more easily.

In this chapter, we are interested in evaluating the size of the performance penalty of using out-of-domain data to train a model for parsing with a precision grammar, and also in ways in which this performance penalty can be alleviated, particularly when a small amount of domain-matched data exists or can be created and we wish to get the most possible value out of this data, or by using a self-training strategy to automatically create lower-quality in-domain training data. We take advantage of the existence of multiple ERG treebanks of sufficient size, and spanning complex and variable domains, enabling us to explore these issues systematically for precision

HPSG parsing. We also outline a methodology for quantitatively and qualitatively comparing different corpora for lexico-syntactic divergences.

4.2 Corpora

From the treebanks described in Section 3.4, there are two corpora available for the ERG which are large enough individually for training parse selection models. The LOGON corpus contains 8550 sentences with exactly one gold parse, which we partitioned randomly by sentence into 10 approximately equal sections, reserving two sections as test data, and using the remainder as our training corpus. In Section 3.4 we noted that LOGON includes translations from Norwegian. In fact, the translations include alternative English translations of each source sentence from up to three different translators (although direct repetitions are very rare). To ensure that the similarities between these translations did not interfere with the results, we placed equivalent translations in the same section, so that two translations of the same sentence would never occur between both training and test corpora. We also use the WeScience corpus, which has 11558 sentences in total. From these, we randomly chose 9632 sentences, preserving the remainder for future work. We partition these sentences in a similar way into 10 equal sections, keeping two as a test corpus. This means that the corpora as we use them here are approximately equal in size in terms of number of tokens. Ideally, we would have used cross-validation here to get more precise measurements from the few thousand sentences we have available, however this would have complicated the experimental design significantly in subsequent experiments where we combine in-domain and out-of-domain data using cross-validation for parameter tuning, and would also have made the experimentation phase far more time-consuming.

We additionally make use of the smaller corpora mentioned in Section 3.4: the Cathedral and Bazaar corpus, and 'robot1'. We give some broad summaries of the corpora in Table 4.1, which is partly repeated from Section 3.4 for convenience. For the parse selection experiments in both training and testing, we only use the 'validated sentences', which are those sentences which are within the coverage of the grammar, and for which the annotator marked a tree as acceptable in the treebanking stage (i.e the annotator didn't reject all trees), so the counts for training and test sentences reflect this.

We show some more details of sentence length in the histogram in Figure 4.1. The distributions are mostly unsurprising, with the longer-tailed distribution for C&B that we would expect for the longer average sentence length. The most noticeable value is the unexpectedly large value in the 0–5 range for WeScience, which is likely a reflection of the frequent occurrence of short article titles in this corpus.

	WeScience	LOGON	C&B	robot1
Total Sentences	9632	9410	770	1537
Parseable Sentences	9249	8799	667	1412
Validated Sentences	7631	8550	567	1303
Train/Test Sentences	6149/1482	6823/1727	0/567	768/535
Tokens/sentence	15.0	13.6	18.7	5.8
Training Tokens	92.5k	92.8k	0	4.5k
Parses per sent	274.1	236.5	322.8	100.7
Random (exact match)	11.7%	13.0%	6.7%	25.9%
Random (top 10)	31.4%	35.4%	21.1%	63.9%

Table 4.1: Corpora we use for our experiments showing numbers of sentences which have a single gold parse (which are the only sentences used in these experiments), average sentence length (in tokens), average ambiguity on test set (number of parses produced per sentence with each sentence capped at 500 parses, approximating, but underestimating, the difficulty of the parse selection problem), and the random baseline calculated as the average probability of randomly selecting the correct parse for each sentence from this top 500 according to the parse selection model used in treebanking.

4.2.1 Detailed Corpus Statistics

In Table 4.1, we have already examined the high-level distinctions between the corpora in terms of the ambiguity of the sentences and the related accuracy we can expect as a random baseline. However, there are several other characteristics which may be of interest for the parse selection problem. Table 4.2 shows selected finer-grained statistics. This includes the total number of word types (i.e. the vocabulary size of the corpus) and of those how many are outside the vocabulary of the grammar version that we are using – giving an indication of how many items the grammar must handle without a native lexical entry. WeScience stands out as having a large vocabulary, as well as a high proportion of unknown types, with 43.4% of types outside the lexicon, which is fairly unsurprising given the nature of its content. LOGON has a lower proportion at only 9.4%, which reflects the history of the grammar – as noted above, it was extensively expanded for the LOGON project, so the lexicon is somewhat tuned to match that corpus. C&B, at only 11.2%, also shows a fairly small percentage of unknown types, while the very small vocabulary of robot1 has, perhaps unsurprisingly, an even lower proportion of 2.3%, although this could also be partly due to grammar tuning. As we would expect, on a per-token basis, the number

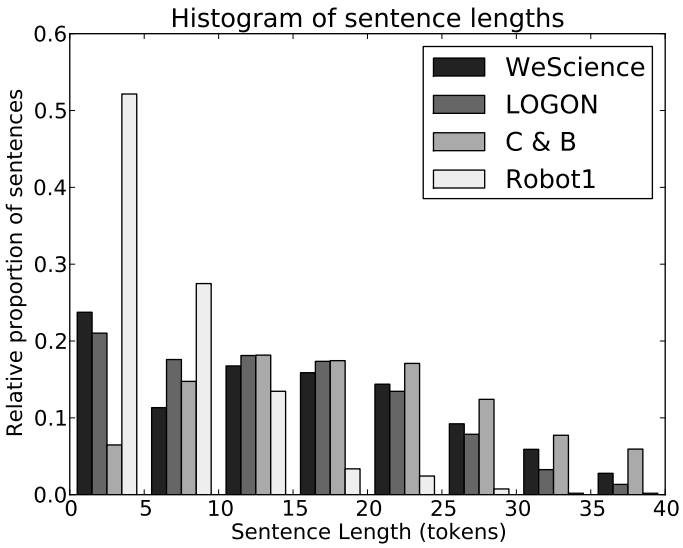


Figure 4.1: Histogram of sentence lengths for the corpora

of unknown words is much smaller, and the relative differences between the corpora also decrease, ranging from 0.3% of tokens for ROBOT1 to 7.0% for WESCIENCE.

Table 4.2 also shows statistics on the applications of ERG rules to give an idea of the relative complexities of the parse trees, separately to the sentence length. We show statistics on a per-sentence as well as a per-token basis, as the latter attempts to factor out the effects of sentence length on the complexity. As we noted in Section 3.1.1, the ERG rules can be divided up into "construction rules", which correspond to unary or binary rules accounting for syntactic phenomena, and "lexical rules", which can be roughly equated to morphological and morphosyntactic phenomena such as inflection.[1] We show a crude comparison of the number of applications of these rules in the gold-standard parse trees in the corpora. Construction rules are not meaningfully different on a per-token level, suggesting that perhaps after scaling for sentence length, the corpora have similar levels of syntactic complexity. However, there are substantially more applications of lexical rules in WESCIENCE and LOGON than in C&B and particularly ROBOT1. Some explanations for this are discussed in Section 4.2.2.

We also show some finer-grained statistics, intended to provide more insight into the distribution of some selected interesting syntactic phenomena which show different distributions between the corpora. WESCIENCE has a much higher incidence of

[1] A third class of rules relates to bookkeeping for punctuation, which is linguistically less interesting and ignored here.

Validated Sentences	7631	8535	567	1303
Word Types	13662	7888	2744	664
O.O.V. Word Types	5931	744	306	15
Tokens	15.02 [1.000]	13.63 [1.000]	18.67 [1.000]	5.82 [1.000]
O.O.V. Tokens	2.02 [0.070]	1.49 [0.025]	1.66 [0.039]	1.11 [0.003]
Construction Rules	20.74 [1.380]	18.85 [1.383]	25.72 [1.378]	8.28 [1.421]
Lexical Rules	5.78 [0.382]	5.13 [0.372]	6.12 [0.322]	2.34 [0.245]
Noun Compounds	1.10 [0.073]	0.58 [0.043]	1.10 [0.059]	0.12 [0.021]
Co-ordination	0.65 [0.043]	0.67 [0.049]	0.56 [0.030]	0.09 [0.015]
Passives	0.54 [0.036]	0.27 [0.020]	0.28 [0.015]	0.02 [0.004]

Table 4.2: Corpora we use for our experiments showing numbers of sentences which have a single gold parse, number of word types (total and out-of-vocabulary), and statistics for various interesting tokens and constituents, both per-sentence and per-token (in square brackets). O.O.V. denotes 'out-of-vocabulary' for the lexicon of the '1010' version of the ERG which we use here.

the passive voice,[2] with 0.036 instances per token indicating almost twice as many occurrences per token than the nearest competitor LOGON with 0.020. This is probably in line with what we would expect given the nature of the different domains, with WeScience containing more formal academic-style prose, rather than the slightly more conversational style of LOGON. C&B has a lower incidence still — again, the style is more conversational, and the essay is written from a first-person perspective. ROBOT1 has by far the lowest, but this is just what we would expect for spoken interactive dialogue.

The relative differences between the corpora of frequency of noun compounds and coordination[3] are smaller, but still noticeable. The technical subject matter of WeScience may partially account for the greater frequency of noun compounds. The coordination differences are not as easy to explain, except for their infrequent occurrence in the simple ROBOT1 sentences.

4.2.2 Inter-corpus Comparisons

It is clear from Section 4.2.1 that the corpora have different characteristics in terms of broad statistical counts, but it may also be informative to directly compare

[2]This was determined by counting instances of grammar rules which are subtypes of BASIC_PASSIVE_VERB_LR.

[3]These are subtypes of BASIC_N_N_CMPND_PHR and BASIC_COORD_PHR respectively.

pairs of corpora to measure how alike they are in a more statistically-grounded way. While we have avoided answering the question of exactly what constitutes a domain, instead equating corpora with domains, a statistical measure of corpus similarity should partially serve as a proxy for this. Counting how many words occur in only one of the corpora gives us some idea of the difference, however this discards most of distributional information. To make a more thorough comparison, we follow the technique of Verspoor *et al.* (2009) in using relative entropy to compare pairs of corpora, which we briefly describe here.

We choose some vocabulary V which is a subset of the union of the vocabularies of the two corpora, and we then construct probability distributions P_1 and P_2 using maximum likelihood estimation and add-one smoothing for each corpus. We can calculate the relative entropy over the words in that vocabulary for corpus 1 against corpus 2 as follows:

$$D(P_1||P_2) = \sum_{w \in V} P_1(w) \log_2 \frac{P_1(w)}{P_2(w)} \tag{4.3}$$

This gives us a way of quantifying how different the distributions of words are between corpora. Also following Verspoor *et al.*, we show values for different frequency cutoffs after sorting the vocabulary by combined frequency between the two corpora.

However, we may also be interested in the words which most strongly characterise the differences between the corpora. Rayson and Garside (2000) outline a way to achieve this using log-likelihood. Given two corpora with total number of tokens T_1 and T_2, we can calculate the expected frequency count for a particular word with observed counts t_1 and t_2 in each corpus as follows:

$$E_1 = \frac{T_1(t_1 + t_2)}{T_1 + T_2} \quad E_2 = \frac{T_2(t_1 + t_2)}{T_1 + T_2}$$

From this, we can calculate the log-likelihood for the particular word:

$$L = 2 \left(t_1 \log_2 \frac{t_1}{E_1} + t_2 \log_2 \frac{t_2}{E_2} \right)$$

If we sort the log-likelihood values for each word by decreasing log-likelihood values, those items at the top of the list are those which are most significantly different between the corpora and thus in one view characterise the differences between the corpora.

However in our work, we are also interested in the distributions of syntactic constructions between the corpora to see whether the differences extend beyond the vocabularies. We can achieve a coarse version of this using a variant of the word-based procedure described above. We take advantage of the fact that the rules in the ERG are named according to their syntactic function. The grammar has 160 construction rules and lexical rules to account for different phenomena. For example,

Ref: WeSc (train)	**WeSc**	LOG	C&B	robot1
Types in ref	50.9%	32.6%	48.7%	51.7%
Tokens in ref	86.7%	63.7%	78.6%	63.3%
Rel-Ent (Top-100 words)	0.00/0.03	0.52/1.14	0.38/0.36	1.04/4.14
Rel-Ent (Top-1k words)	0.11/0.14	2.09/2.70	0.96/1.61	1.71/7.45
Rel-Ent (Top-10k words)	0.44/0.60	3.05/3.34	0.73/3.32	0.75/10.56
Rel-Ent (all rules)	0.01/0.01	0.33/0.34	0.27/0.28	1.82/2.94
Rel-Ent (constructions)	0.01/0.01	0.29/0.31	0.28/0.28	1.99/3.23
Rel-Ent (lex rules)	0.00/0.00	0.46/0.42	0.19/0.23	0.94/0.95
Ref: LOG (train)	WeSc	**LOG**	C&B	robot1
Types in ref	22.6%	54.3%	31.7%	57.1%
Tokens in ref	59.2%	87.6%	65.6%	69.9%
Rel-Ent (Top-100 words)	1.25/0.51	0.02/0.03	1.08/0.49	1.06/3.25
Rel-Ent (Top-1k words)	2.72/2.01	0.25/0.24	2.11/2.11	1.76/6.28
Rel-Ent (Top-10k words)	3.25/3.02	0.69/0.65	1.95/4.11	1.20/8.20
Rel-Ent (all rules)	0.35/0.34	0.01/0.02	0.43/0.40	1.60/2.68
Rel-Ent (constructions)	0.32/0.29	0.01/0.02	0.40/0.35	1.85/2.98
Rel-Ent (lex rules)	0.47/0.50	0.01/0.01	0.53/0.59	0.45/0.62

Table 4.3: Test corpora compared with reference training corpora (1833-sentence subset of WeScience; 1710-sentence subset of LOGON). The in-domain test set (i.e. a different subset of the same source corpus) is labelled in bold. Relative entropy is shown for reference corpus against test corpus and then vice versa, and uses add-one smoothing, only examining items with at least 5 occurrences (10 for grammatical rules) in the combination of the corpus pair.

three subject-head rules account for the subject of a sentence attaching to the verb in different circumstances, while twelve different rules are used for different kinds noun-compounding. Like lexical items, these rules have widely different relative frequencies of use from one another, and we also expect these rules may have different frequencies of usage between different corpora. By replacing words in the procedures described above with rule names, so that the vocabulary is the complete inventory of available rules, we can calculate relative entropy figures and log-likelihood figures across lexical rules, construction rules, or a combination of both. In fact Rayson and Garside (2000) noted that their procedure could be applied to other entities such as POS-tags, so applying it to syntactic construction names is within the original spirit of the method.

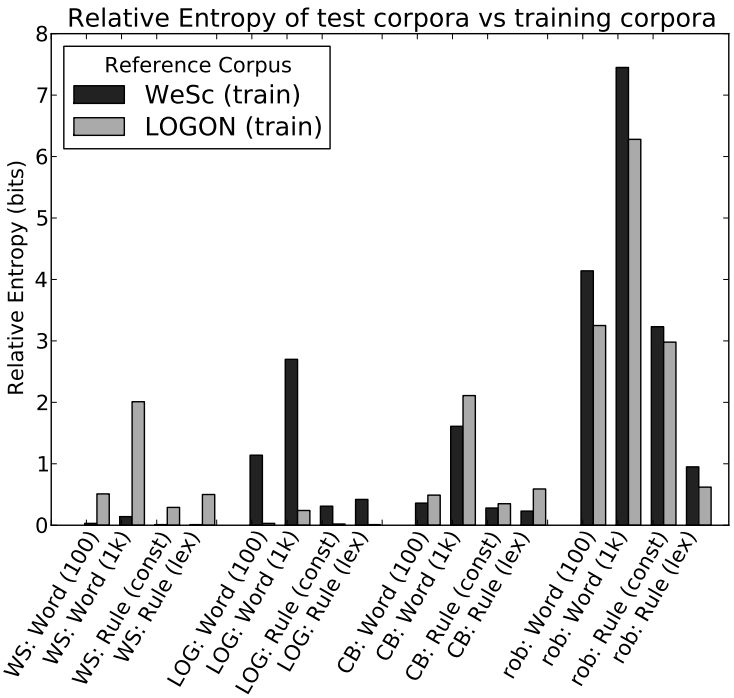


Figure 4.2: Graphical representation of relative entropy of the test corpora against the two reference corpora (subset of the information in Table 4.3)

We could possibly improve over aggregating solely by rule name – some very general rules such as the head-complement rules can apply to a wide range of lexical heads with complement slots, including prepositions, verbs, adjectives and nouns, and we are ignoring this potentially interesting information. Nonetheless, this fairly simple procedure provides a concise indication of the relative syntactic similarity between the corpora.

We show the results for relative entropy against subsets of each training corpus in Table 4.3, and in graphical format in Figure 4.2. The relative entropy differences between the in-domain comparison versus out-of-domain comparison are striking. The in-domain relative entropy is 0.03 or less for the top-100 words and 0.25 or less for the top-1000, versus 0.36 or more and 0.96 or more for the corresponding out-of-domain figures. It is perhaps not so surprising that we should see low relative entropy on a per-word basis within a domain, and high relative entropy on a word-by-word basis compared to other corpora with different subject matter and register, where

WeSc (tr) vs LOG	WeSc (tr) vs C&B	WeSc (tr) vs robot1
you[800.9]→	i[799.2]→	okay_s_adv[960.7]→
trail_n1[575.3]→	bazaar_n1[249.4]→	box_n1[701.0]→
←[552.2]language_n1	←[200.6]language_n1	i[655.9]→
trip_n1[529.3]→	you[165.2]→	yeah_root_pre[534.8]→
mountain_n1[435.6]→	fetchmail_n1[157.5]→	you[533.2]→
route_n1[428.6]→	project_n1[140.2]→	room_n1[451.5]→
lodge_n1[418.8]→	"Linus"[137.9]→	go_v1[331.6]→
mark_v1[385.5]→	mail_n1[132.7]→	uh_disc_adv[307.0]→
glacier_n1[382.0]→	me[125.1]→	blue_a1[297.3]→
go_v1[358.4]→	bug_n1[125.1]→	um_disc_adv[297.1]→
down_adv1[346.9]→	my[123.7]→	alright_root_pre[287.2]→
road_n1[340.6]→	open_a1[121.1]→	door_n1[264.8]→
hike_n1[331.4]→	cathedral_n1[118.1]→	hallway_n1[245.3]→
along[321.0]→	development_n1[108.3]→	be_th_cop_is_cx_2[235.5]→
to[279.9]→	your[108.1]→	get_v2[223.9]→
summer_n1[271.5]→	developer_n1[94.6]→	yes_pre_root[210.8]→
hike_v1[271.5]→	do2[88.4]→	be_c_am_cx_2[208.0]→
up_adv1[271.0]→	it2[85.3]→	turn_v1[188.2]→
summit_n1[253.1]→	linux_n1[82.5]→	be_id_is_cx_2[178.9]→
←[252.6]system_n1	←[80.5]word_n1	green_a1[176.6]→
at[249.8]→	get_prd_v1[75.2]→	left_n1[169.3]→
to_nmod[244.5]→	we[73.7]→	chair_n1[159.7]→
here_nom[235.7]→	think1[73.3]→	there_expl[153.3]→
side_n1[226.0]→	design_n1[72.7]→	that_det[151.7]→
hut_n1[220.9]→	hacker_n1[72.2]→	so_adv1[139.9]→
valley_n1[220.9]→	debug_v1[72.2]→	corner_n1[138.7]→
←[215.9]data_n2	source_n1[71.4]→	there_nom[132.7]→
stone_n2[211.7]→	law_n1[69.2]→	where_nom[125.7]→
tour_n1[211.7]→	brooks_n1[68.9]→	number_n3[123.9]→
bygdin[202.5]→	←[67.0]data_n2	pink_a1[118.8]→

Table 4.4: Lexical entries with highest log-likelihood against WeScience training corpus as reference. '←' denotes a higher frequency in the reference corpus; '→' denotes a higher frequency in the test corpus

LOG (tr) vs WeSc	LOG (tr) vs C&B	LOG (tr) vs robot1
language_n1[886.2]→	i[784.4]→	okay_s_adv[965.5]→
←[484.8]you	linux_n1[310.7]→	box_n1[712.5]→
data_n2[471.8]→	software_n1[304.0]→	i[643.8]→
software_n1[361.1]→	project_n1[266.7]→	yeah_root_pre[537.5]→
system_n1[354.4]→	bazaar_n1[251.2]→	room_n1[392.4]→
algorithm_n1[293.7]→	development_n1[204.7]→	uh_disc_adv[308.5]→
model_n2[284.0]→	source_n1[194.8]→	um_disc_adv[298.6]→
←[264.6]trip_n1	developer_n1[191.7]→	alright_root_pre[288.6]→
←[262.2]route_n1	design_n1[185.1]→	door_n1[266.2]→
←[261.6]trail_n1	mail_n1[171.9]→	hallway_n1[258.8]→
←[256.8]mountain_n1	user_n1[171.9]→	blue_a1[240.6]→
computer_n1[240.7]→	"Linus"[171.9]→	yes_pre_root[226.9]→
word_n1[238.4]→	internet_n1[158.6]→	be_c_am_cx_2[197.4]→
←[232.2]to	fetchmail_n1[158.6]→	number_n3[169.2]→
use_v1[228.2]→	me[152.0]→	that_det[162.9]→
←[226.6]lodge_n1	bug_n1[152.0]→	chair_n1[154.6]→
application_n1[192.6]→	code_n1[152.0]→	get_v2[142.0]→
network_n1[190.7]→	problem_n1[142.7]→	corner_n1[128.8]→
machine_n1[187.8]→	←[135.1]trip_n1	one_adj[122.3]→
of_prtcl[185.2]→	←[133.6]trail_n1	pink_a1[119.4]→
math_xml_n1[182.9]→	←[131.1]mountain_n1	me[109.5]→
speech_n1[182.9]→	community_n1[127.4]→	minute_n1[109.5]→
information_n1[180.3]→	law_n1[120.9]→	another[102.8]→
←[179.0]along	←[120.1]route_n1	green_a1[101.7]→
←[178.2]up_adv1	cathedral_n1[119.0]→	left_n2[99.6]→
←[172.7]go_v1	←[115.7]lodge_n1	okay_s_adv3[99.5]→
←[170.6]glacier_n1	model_n2[112.4]→	ok_s_adv[99.5]→
program_n1[168.5]→	my[112.2]→	left_n1[96.2]→
←[167.6]down_adv1	open_a1[90.7]→	turn_v1[94.9]→
english_n1[163.7]→	←[89.9]mark_v1	where_nom[94.9]→

Table 4.5: Lexical entries with highest log-likelihood against LOGON training corpus as reference. '←' denotes a higher frequency in the reference corpus; '→' denotes a higher frequency in the test corpus

WeSc (tr) vs LOG	WeSc (tr) vs C&B	WeSc (tr) vs ROBOT1
np_adv_c[692.5]→	hdn_bnp-qnt_c[754.3]→	root_spoken_frag[2636.4]→
←[646.7]hdn_bnp_c	←[267.8]hdn_bnp_c	r_scp-frg_c[1482.2]→
hdn_bnp-pn_c[570.9]→	←[228.9]root_frag	hdn_bnp-qnt_c[761.2]→
←[524.6]hdn_optcmp_c	←[209.3]np_frg_c	←[624.5]hdn_bnp_c
hdn_bnp-qnt_c[395.9]→	sb-hd_nmc_c[205.7]→	aj-hd_scp_c[583.8]→
num-n_mnp_c[299.9]→	cl_cnj-frg_c[201.8]→	root_informal[552.8]→
hd-aj_int-unsl_c[289.6]→	←[199.4]hdn_bnp-pn_c	hd_imp_c[398.9]→
←[278.2]aj-hdn_norm_c	hd_xcmp_c[96.2]→	←[368.4]hdn_bnp-pn_c
n_sg_ilr[1462.1]→	v_pst_olr[235.0]→	v_n3s-bse_ilr[337.0]→
←[1120.0]n_ms-cnt_ilr	v_n3s-bse_ilr[216.8]→	v_aux-sb-inv_dlr[263.1]→
←[190.9]v_nger-tr_dlr	←[117.8]v_pas_odlr	←[104.2]n_ms-cnt_ilr
←[152.8]w_italics_dlr	←[69.9]n_pl_olr	←[97.3]v_pas_odlr
←[141.3]v_pas_odlr	←[59.2]w_italics_dlr	←[77.8]n_pl_olr
v_nger-pp_dlr[132.7]→	v_np-prtcl_dlr[45.3]→	n_sg_ilr[49.4]→
←[93.0]v_prp_olr	v_aux-sb-inv_dlr[43.2]→	v_pst_olr[47.8]→
←[66.8]v_pas-dat_odlr	v_psp_olr[39.5]→	←[44.3]v_nger-tr_dlr

Table 4.6: Construction rules (first section, ending in '_c') and lexical rules (ending in 'lr') with the highest log-likelihoods against WeScience training corpus as reference

we expect a widely different vocabulary (as the figures for token-overlap with the reference corpus indicate).

In Tables 4.4 and 4.5, we show the words with the greatest log-likelihood differences between the corpora. We can see some instructive differences between the corpora (noting that items prefixed with '←' are more frequent in the reference corpus, and those followed by '→' are more frequent in the test corpus). C&B and ROBOT1 are characterised by much more frequent use of the pronoun *I* than the other corpora, which is predictable as C&B is a conversational first-person essay and ROBOT1 is a transcription of dialog. The second-person pronoun *you* is somewhat characteristic of all the corpora (less strongly for C&B) compared to WeScience, again indicative of the styles we would expect – the detached formal prose of WeScience very rarely uses first and second person pronouns. The other more specific vocabulary items are also clearly characteristic of the domain – it is unsurprising that C&B talks about *bazaars* and *software* and LOGON talks about *trails* and *routes* while WeScience talks about *language* and *data*.

Perhaps more interesting and surprising than the distributions of lexical items is the different distributions of syntactic constructions and lexical rule applications.

LOG (tr) vs WeSc	LOG (tr) vs C&B	LOG (tr) vs robot1
hdn_bnp_c[877.3]→	←[780.2]hdn_bnp-pn_c	root_spoken_frag[2642.7]→
←[478.6]hdn_bnp-pn_c	hdn_bnp-qnt_c[180.5]→	r_scp-frg_c[1528.4]→
←[459.6]np_adv_c	←[159.5]root_frag	←[666.1]hdn_bnp-pn_c
aj-hdn_norm_c[393.1]→	hdn_optcmp_c[148.2]→	aj-hd_scp_c[593.7]→
hdn_optcmp_c[388.8]→	←[142.5]np_frg_c	root_informal[355.4]→
←[319.6]hdn_bnp-qnt_c	that_c[125.0]→	hdn_bnp-qnt_c[348.0]→
n-hdn_cpd_c[283.9]→	hd_xcmp_c[118.0]→	←[239.8]hdn_bnp_c
vp_rc-redrel_c[276.7]→	←[117.6]hd-aj_int-unsl_c	←[182.9]root_strict
n_ms-cnt_ilr[1370.4]→	←[678.0]n_sg_ilr	v_aux-sb-inv_dlr[269.8]→
←[1296.6]n_sg_ilr	n_ms-cnt_ilr[534.0]→	v_n3s-bse_ilr[241.4]→
v_nger-tr_dlr[180.4]→	v_pst_olr[177.7]→	←[41.2]v_pas_odlr
v_pas_odlr[178.2]→	v_prp_olr[114.1]→	←[38.8]n_pl_olr
w_italics_dlr[140.3]→	v_n3s-bse_ilr[94.1]→	v_pst_olr[35.2]→
v_prp_olr[112.3]→	v_v-re_dlr[63.8]→	←[31.9]v_3s-fin_olr
n_pl_olr[80.4]→	v_aux-sb-inv_dlr[47.2]→	←[28.2]n_sg_ilr
n_ms_ilr[78.1]→	v_nger-tr_dlr[38.3]→	←[16.9]j_att_dlr

Table 4.7: Construction and lexical rules with the highest log-likelihoods against LOGON training corpus as reference

While these differences are much less dramatic in terms of bits of entropy, with out-of-domain corpora having as few as 0.19 bits of relative entropy compared to in-domain, there are still noticeable differences between the corpora, and it is not necessarily clear that this should be the case. While we might suspect, for example, that a dialogue-based corpus like robot1 would have a higher percentage of imperative constructions, we could not know whether this would have a noticeable impact on the overall distribution. In fact, robot1 stands out as being very different syntactically, but we can also see some other noticeable differences between the corpora. C&B seems to be most similar to WeScience in terms of syntax as well as vocabulary.

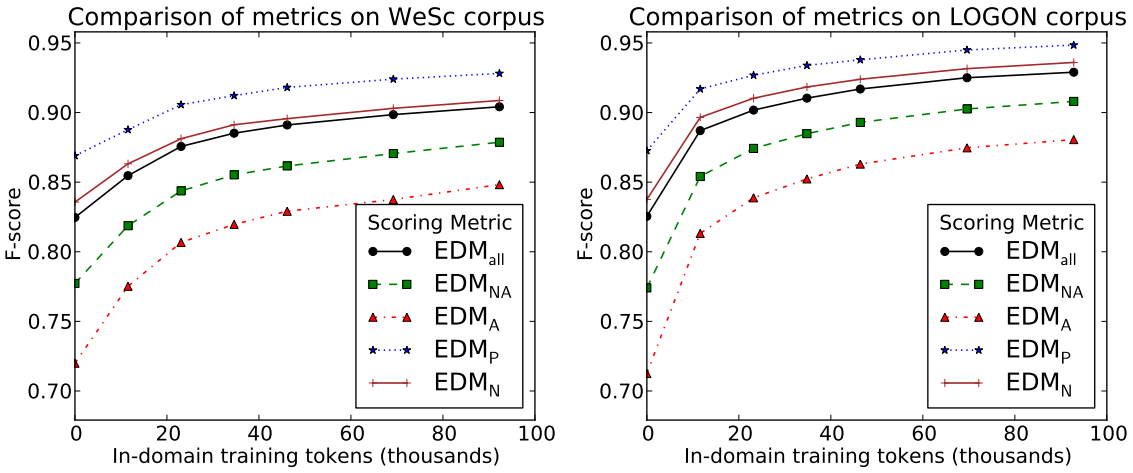
Examining the details of the rules that characterise the inter-corpus differences shown in Tables 4.6 and 4.7, they are less easy to explain than the lexical items, even with knowledge of the meanings of the rules within the ERG internals. WeScience has a disproportionately high number of occurrences of hdn_bnp_c, corresponding to noun phrases headed by common nouns with no determiners, such as *algorithms* possibly due to article titles and factual, general subject matter. Meanwhile LOGON has a preponderance of hdn_bnp-pn_c instances, denoting bare noun phrases with proper nouns, like *Jotunheimen*, a region in Norway, which is probably related to the

large number of place names mentioned. Another unusually frequent construction in LOGON is NP_ADV_C, for adverbial noun phrases like *here* in *They came here*. This discrepancy between proper noun usage may also partially explain the much higher prevalence of HDN_OPTCMP_C in WeScience, which is used for common nouns with optional complements. A cursory examination of the parse trees suggests that another contributing factor is the fact that nouns with optional complements are often nominalisations such as *implementation*, and these constructions seem particularly frequent in the technical subject matter of WeScience.

Amongst the lexical rules, there are several that stand out as having very different distributions between the corpora. N_SG_ILR is strongly characteristic of LOGON against most comparison corpora, while N_MS-CNT_ILR is much more frequent in WeScience and C&B. This is probably due to the higher proportion of tokens in WeScience and C&B which are outside the grammar's lexicon — as shown in Table 4.2, this is 7% and 3.9% of tokens respectively, compared with 2.5% for LOGON. For the large proportion of unknown words which are POS-tagged in preprocessing with the TnT tag 'NN', the word is translated into a generic lexical entry which is agnostic about the mass or count status of the noun (as this cannot be determined from the POS tag), and N_MS-CNT_ILR rule is then applied to the resulting token by the grammar. It is also used by a large number of actual lexical entries which can genuinely be mass or count nouns, such as STRUCTURE_N1, and these may also be more common in the WeScience domain, but the preprocessing differences are likely to be more important here. So, this is more of an artefact of the grammar history and the preprocessing than a reflection of underlying linguistic differences, but nonetheless, still reflect differences that will affect the parse selection process.

There are also some less strongly-distinguished lexical rules. The higher frequency of V_PAS_ODLR in WeScience, corresponding to passive constructions, is unsurprising given the higher frequency of passives noted in Table 4.2. WeScience also has a high prevalence of V_NGER-TR_DLR, which is applied to gerunds such as *processing*. As noted above, nominalisations are frequent in this technical domain, so it is unsurprising that gerunds, which are a subtype of nominalisation, are also frequent. Finally, we can see that V_PST_ODLR, for past tense verbs, is characteristic of C&B. This makes sense, since C&B is more concerned with the author's personal historical narrative.

From all of this, we can see some reasons why a parse selection model trained in one domain may perform poorly over a second domain. Not only are there differences in distributions of lexical items to take into account, there are also widely different distributions of rule applications, both lexical and syntactic, which can all correspond to features in the parse selection model.



(a) WeScience (b) LOGON

Figure 4.3: Comparison of variants of the EDM metric

4.3 Evaluation

For these parse selection experiments, we use a selection of the evaluation metrics described in Section 3.6. Specifically, we use Acc_1 and Acc_{10}, to give the best possible measures of treebanking utility, and EDM to provide a more granular evaluation that may give a better indication of performance in downstream tasks.

4.3.1 Empirical Comparison of EDM Configurations

As detailed in Section 3.6.3, there are several different classes of semantic information that we can choose to include or exclude in the EDM evaluation: ARGS, NAMES and PROPS. We have not yet evaluated empirically how the results differ for the different combinations of classes. In Figure 4.3, we show the learning curves that result from training each of our test corpora on different volumes of in-domain training data from the corresponding training corpora across various EDM configurations. We show results for each class individually (EDM_A, EDM_N and EDM_P for ARGS, NAMES and PROPS respectively) as well as the combined results including triples from all three classes (EDM_all). We also show results for EDM_NA, excluding PROPS only, which we argued in Section 3.6.3 was best for cross-framework comparability. We could also have applied different weightings to the triples in each class, but for simplicity here we only show the results for inclusion and exclusion, corresponding to weights of either one or zero for each class.

Here and below, we create the zero-training data points on the y-axes for EDM by randomly selecting a parse from the top-500 (at most) parses included in the treebanks distributed with the ERG, and choosing one of those at random to create the EDM triples, repeating the process over the whole test corpus 10 times to get an average.

From Figure 4.3, we can see from any vertical slice that different classes of triples are harder or easier to determine correctly. The PROPS triples of EDM_P give the highest score, which is largely due to the fact that this information class is based on many characteristics which are relatively easy to determine. An important reason for this is that these features are strongly inter-dependent, and the grammar is highly constrained, which means that many implausible candidates for the feature values or combinations of feature values are never even considered by the grammar for any candidate parse, so are correct even when we only have a very poor parse selection model. A related secondary factor is that some features have a very low information content and high majority class baseline. For example [PERS 3] for third person occurs on almost all common nouns, so if we simply inserted the triple corresponding to [PERS 3] for every nominal entity (including first and second person pronouns, where it would be incorrect), we would get a substantial contribution to the F-score. The grammar is of course more carefully crafted than this, and deliberately sets the correct value for PERS on each lexical entry through inheritance, so that it does indeed get the correct value for the rarer case of first and second person pronouns, but this is not reflected by a bulk metric such as this. EDM_N is also fairly high, showing that the grammar generally postulates the correct predicate name, which means that it is at a minimum applying the correct part of speech and verb subcategorisation in most cases. This is again something that can be achieved reasonably easily in a large percentage of cases purely from the textual string of the corresponding token, but any gains the grammar has over this baseline would also not be reflected.

It also appears that the different EDM metrics are very strongly correlated. This is least surprising when one set of triples is a proper subset, and includes a large proportion of the same elements, such as for EDM_{NA} and EDM_{all}, which differ only by the inclusion of the fairly consistent PROPS class, and have very high pairwise Pearson correlation (as calculated from these graphs) of 0.9996. However, the correlation also holds fairly strongly for the non-overlapping triples of EDM_A, EDM_N and EDM_P alone; the correlations between each possible pairing of these are between 0.9958 and 0.9986, which still show very strong relationships. Again, this is partially a reflection of interdependencies between the triples as a results of the constrained grammar – if the grammar gets most dependencies correct as reflected in the ARGS class, it gets most triples from the PROPS class as well.

This strong correlation suggests that there is little value in reporting multiple different variants of EDM for each experiment – on average, changes in one variant will show up in the other, although this may not be true on the level of individual sentences, and the overall magnitude of the change will be different. We instead

	WeScience				LOGON			
Length	Sents	Acc_1	Acc_{10}	EDM_{NA} P / R / F	Sents	Acc_1	Acc_{10}	EDM_{NA} P / R / F
0–4	350	87.4	99.7	94.8 / 94.7 / 94.8	361	93.1	100.0	97.9 / 97.8 / 97.8
5–9	167	62.3	94.6	92.4 / 91.6 / 92.0	302	71.2	96.0	95.3 / 95.2 / 95.3
10–14	247	46.6	88.7	93.5 / 91.7 / 92.6	311	58.5	90.0	95.5 / 95.6 / 95.6
15–19	234	26.5	65.4	91.5 / 89.6 / 90.6	298	34.6	71.8	94.5 / 91.8 / 93.1
20–24	212	20.3	61.8	92.6 / 90.4 / 91.5	231	20.3	58.9	93.1 / 91.9 / 92.5
25–29	136	12.5	36.8	91.3 / 87.1 / 89.1	135	11.1	31.1	92.8 / 92.6 / 92.7
30–34	87	6.9	27.6	92.3 / 90.7 / 91.5	56	10.7	32.1	93.7 / 89.4 / 91.5
35–39	36	2.8	16.7	91.7 / 89.8 / 90.8	21	4.8	14.3	93.3 / 84.6 / 88.8
40–44	11	0.0	9.1	89.3 / 72.4 / 80.0	7	0.0	0.0	93.1 / 81.5 / 86.9
45+	2	0.0	50.0	98.5 / 36.5 / 53.3	5	20.0	20.0	97.5 / 98.4 / 98.0
ALL	1482	44.1	73.7	89.2 / 86.5 / 87.9	1727	52.5	77.9	91.6 / 90.0 / 90.8

Table 4.8: WeScience and LOGON test corpora using models trained on all in-domain tokens aggregated by sentence length, scored against exact match (Acc_1), top-10 match (Acc_{10}) and EDM_{NA}

pick one variant as primary. It seems reasonable that we should include ARGS, since this reflects the relationships between the entities in the sentence, most closely corresponding the core dependencies used elsewhere. The NAMES class is also important, since it shows information about whether we are accurately mapping the tokens to predicates. This leaves EDM_{NA} and EDM_{all} as options. There are reasons to include or exclude PROPS — including it gives credit for accurately predicting properties that are in fact available in the grammar, but on the other hand, it tends to inflate performance figures due to the relative ease with which the PROPS triples can be predicted compared to the other classes. EDM_{NA} also has the advantage of being slightly more comparable with other formalisms as noted in Section 3.6.3 (although still not directly comparable), so it is the primary EDM variant we use to report results.

4.3.2 Empirical Comparison of Exact and Granular Metrics

It is also worth considering at this point whether the differences we could observe between the behaviours of exact match and dependency-based evaluation are purely because the EDM metric is far less sensitive to effects of sentence length. Correcting Acc_1 for sentence length is difficult, but we show in Table 4.8 the various results aggregated by sentence length trained on solely in-domain data. When comparing similar sentence lengths, higher Acc often correlates with higher EDM_{NA}, although the

opposite is seen in some cases, even for a well-populated range like 25–29, indicating that EDM is giving us information beyond what we get from Acc adjusted for sentence length.

Relatedly, we can see that LOGON gets higher exact match performance over a similar quantity of training data than WeScience – we would like to know if this is purely because of the slightly higher sentence length and average ambiguity of WeScience. The lower EDM score for in-domain WeScience compared to in-domain LOGON in Figure 4.3 makes us suspect that WeScience is intrinsically a 'harder' corpus to parse with the ERG (which would be unsurprising, given that it has been more extensively tuned over LOGON), but we would like to evaluate this against Acc as well. The results in Table 4.8 lend some support to the greater difficulty of WeScience, even for sentences of similar length, but mostly over the shorter sentences — although these shorter sentences are also the most heavily populated ranges which will give more reliable data. For all sentence length groups from 0 to 20, LOGON has higher scores than WeScience across all metrics. For the longer sentences, the results are slightly more mixed. WeScience scores higher for Acc_1 in the 25–29 range, for Acc_{10} in the 20–24 and 25–29 ranges, and for EDM in the 35–39 range (although the latter may be too sparsely populated for reliable statistics), but the overall trend is still for LOGON to score more highly on all metrics, independently of sentence length.

Splitting results by sentence length also allows us to more closely examine the EDM numbers. For both corpora in Table 4.8, we see a preference for precision over recall when measured over all sentences, a trend that is repeated in all EDM results reported here. When broken down by sentence length, we see that this trend is fairly stable, but more pronounced as the sentences get longer. Error analysis at the individual sentence level has shown us that this imbalance reflects a slight but genuine tendency of the parser towards simpler analyses with less dependencies, which is perhaps an artefact of the parse selection models being designed to produce the correct syntactic analysis, rather then being optimised to produce semantic dependencies.

4.4 Experiments

The test and training corpora are all originally parsed with PET (Callmeier 2000) and the '1010' version of the ERG. The text was pre-POS-tagged with TnT (Brants 2000) to enable handling of unknown words, and other preprocessing is done using the default settings for the corpus as configured in the [incr tsdb()] performance and competence profiler (Oepen 2001).

We evaluate over corpora in several different domains, training discriminative parse selection models based on manually annotated treebank data drawn from the corresponding training corpus.

In our experiments we are interested first in the effects of domain on parse selection accuracy, as evaluated using our various evaluation metrics (exact match at varying top-N beam widths, and EDM F-score). The second stage of our experiments is designed to investigate the domain adaptation problem, by evaluating different methods of combining in- and out-of-domain data, and to explore how much in-domain data is required to make an appreciable difference to parsing accuracy in a particular domain. Both stages of the experiment drew from the same training and test data sections.

To generate training and test data from the two larger corpora (LOGON and WeScience), we first shuffled the items in each corpus with no regard for section boundaries,[4] and then selected roughly equal-sized training and test item sets from each (see Table 3.4 for the exact item numbers). Finally, we randomly divided the respective training sections into fixed subsections to use in generating learning curves, ensuring there were approximately equal numbers of tokens in the sections from each training corpus (Table 3.4 shows this also corresponds to a similar number of rule applications, since WeScience and LOGON both have 1.38 construction rules per token and around 0.38 lexical rules). The same combinations of training data subsections for a given amount of training data were used in all experiments.

We train the discriminative parse selection models in the framework of Velldal (2007), along the lines of previous work such as Zhang *et al.* (2007) which is described in more detail in Section 3.3.5. This involves feeding in both correct and incorrect parses licensed by the grammar to the TADM toolkit (Malouf 2002), and learning a maximum entropy model. In all experiments, we use the default feature function sets from previous work, with training parameters selected from the grid search which we describe in Section 4.4.1.

The experiments differ in which test data we use out of the four possible test corpora, and in which subset of the 16 sections of the training data we use to train theMaxEnt model. Most simply, we created in-domain and cross-domain learning curves. We trained on 1, 2, 3, 4, 6 or 8 sections of WeScience, and evaluated against all four test corpora, giving one in-domain learning curve (WeScience as test) and 3 cross-domain curves. We also performed the same task using subsets of LOGON as training data instead.

It is also interesting to compare this with the learning curve obtained from mixed-domain training data, to see how the combinations of domains affect the performance on partially in-domain, or cross-domain test data. The simplest variant of this is to train on equally-sized subsets of each. We evaluated all four test corpora against models trained on 2, 4, 6 or 8 pairs of training sections with each pair containing one WeScience section and one LOGON section.

[4]LOGON, however, contains either 2 or 3 English translations of each original Norwegian sentence (as noted in Section 4.2), so as part of this, we ensured that sentences translated from the same source sentence were placed together in the partitioning, to limit the chance of having very similar sentences in multiple sections.

As stated earlier, this has implications for both downstream applications (how confident can I be of providing an accurate top-N parse to end users in a new domain?), and treebanking (how aggressively can I restrict the top-N analyses while still guaranteeing that the correct parse is going to be present?).

To perform the tasks described, we customised a C++ implementation of the feature extraction code and model training code, known as MUPS (duplicating functionality of [incr tsdb()] in some cases). This code is now available to the community as part of the previously-mentioned DELPH-IN code distribution.

4.4.1 Grid Searching to Optimise Training Parameters

To train our models, it is necessary to select a number of parameters in advance. Firstly, we need to set the standard maximum entropy parameters required by TADM: absolute tolerance, relative tolerance and variance (apart from this we use the TADM defaults — in particular the LMVM algorithm described in Section 2.3.1 for learning the feature weights). Additionally, we need to determine optimal values for the *grandparenting level* and *relevant count threshold* described in Section 3.3.5.

While we could have selected these parameters on the basis of solid performers in prior work such as Zhang *et al.* (2007), most of this work assumes a relatively large training set, which is not always the case for our work here; for example, learning curves by their nature require a smaller data set. Velldal (2007) details performing a grid search to optimise these parameters. Essentially we can enumerate all possible values we wish to test for each parameter, and evaluate the performance of each possible combination of those parameters. We can achieve this using cross-validation over our existing corpus, repeatedly holding out a different subset to test on, and training on the remainder then aggregating the results over all the folds. TheMaxEnt training parameters and the model selection parameters can be optimised simultaneously, since there may be interactions between them. For training, we make the reasonable approximation that relevance counts over $\frac{7}{8}$ of the corpus that we use in 8-fold cross-validation are approximately the same as those over the whole corpus, to avoid the need for multiple expensive feature extraction operations for each fold. The optimal parameter combinations are then simply those with the highest exact match accuracy over the training data.

Here, we would ideally like to find the set of parameters which performs optimally over each different set of training data, which is dozens of different training sets. To perform grid searches over all of these, and have different parameters configured for all of these would be prohibitively expensive and complex. Instead, we aimed to find a single set of parameters which performed close to optimally in a range of configurations, from a small single corpus training set to a larger training set encompassing both corpora.

We performed a grid search over the following training corpus combinations, to make sure that we were testing over a representative range of corpus sizes and combinations reflecting the diverse corpus sizes we would be using for training data:

- 11,000-token subsets of both WeScience and LOGON (750–850 sentences)
- 23,000-token subsets of WeScience, LOGON and a 50/50 split of the two
- 46,000-token subsets of WeScience, LOGON and a 50/50 split of the two
- 92,000 tokens of a 50/50 WeScience/LOGON split
- Full WeScience and LOGON training corpora (185,000 tokens/13,000 sentences)

For each corpus combination, we tested 288 different parameter combinations (all possible combinations from those listed below), trying a range of values for each parameter comfortably spanning those which produced near state-of-the-art performance in Zhang *et al.* (2007). We aggregated accuracy over the cross-validation folds to give an overall accuracy figure for each parameter combination, and considered only combinations that ranked in the top 100 out of 288 for all of the corpus combinations (suggesting they are robust across different training data sizes), giving 22 parameter combinations. From these, we selected the combination with the highest mean rank, although all 22 combinations had exact match accuracy within 0.8% of the best for the corpus, so in practice almost any would perform well. For tractability, we followed the standard practice of Velldal (2007) in using a pre-parsed corpus of the top 500 parse trees for each sentence according to some previous model. In each iteration we reranked these parse trees using the new model rather than exhaustively reparsing, which saves a considerable amount of computation time and produces approximately equivalent accuracy figures.

We tested the following parameters in different combinations (the parameters we ultimately selected using the described procedure are highlighted in bold):

- Relevance count threshold: 1, 2, 3, **5**
- Grandparenting level: 1, 2, **3**, 4
- MaxEnt relative tolerance: 10^{-6}, 10^{-8}, $\mathbf{10^{-10}}$
- MaxEnt variance: 10^{4}, 10^{2}, 10^{0}, 10^{-2}, $\mathbf{10^{-4}}$, 10^{-6}

The parameters selected differ from Zhang *et al.* (2007) in using only 3 levels of grandparenting rather than 4, which is unsurprising given the comments above about data-sparseness in smaller training corpora. Two of these parameters are at the extremes of the ranges tested however we kept them since they have performed

Test	Train	Acc_1	Acc_{10}	EDM_{NA}		
				P	R	F
LOG	–	13.0	35.4	79.2	75.7	77.4
	WeSc	36.7[4.5]	66.7[3.1]	86.2[0.9]	83.7[1.1]	84.9[0.6]
	LOG	52.5[2.6]	77.9[3.1]	91.6[0.7]	90.0[1.2]	90.8[0.7]
WeSc	–	11.7	31.4	79.6	76.0	77.7
	WeSc	44.1[2.9]	73.7[3.6]	89.2[1.1]	86.5[2.1]	87.9[1.5]
	LOG	33.6[4.4]	62.4[3.7]	84.3[0.8]	80.8[1.7]	82.5[1.2]
C&B	–	6.7	21.1	80.5	76.5	78.4
	WeSc	27.2[6.0]	61.9[5.0]	88.3[1.2]	84.5[2.8]	86.4[1.7]
	LOG	27.7[3.7]	57.7[7.4]	87.2[0.9]	82.9[2.9]	85.0[1.7]
robot1	–	25.9	63.9	74.0	66.1	69.8
	WeSc	43.4[6.7]	86.4[3.3]	82.3[3.0]	75.3[5.2]	78.6[4.1]
	LOG	45.6[5.7]	89.2[2.3]	82.6[3.9]	70.1[4.4]	75.9[3.7]

Table 4.9: Results using different metrics (exact match on highest-ranked tree [Acc_1], exact match within top-10 [Acc_{10}], and Precision, Recall and F-score over EDM_{NA}), for each of the test corpora, training on pure WeScience or pure LOGON data, or for a baseline randomly selected from the top-500 parses ranked according to a combined WeScience/LOGON model. As in all results, WeSc and LOG test data is 2 sections of held-out data not overlapping with training data. Bold indicates the training data has the same domain as the test data. Figures in square brackets are standard deviations, calculated over 10 random partitions of the data

solidly in previous work, and as noted above, the differences we observed across the range were small, so extending the ranges of parameters tested would be unlikely to produce any improvement.

4.4.2 The Cross-Domain Performance Penalty with Single Domain Models

To evaluate the cross-domain effect, we use the parameters found during the grid search and train parse selection models over differing amounts of training data from LOGON and WeScience. We apply them both in-domain (to the corresponding held-out test data) and cross-domain (to the test datasets for the other corpora) and look at the relative differences between domains, according to our various evaluation metrics. We also created learning curves using different-sized subsets of both We-

SCIENCE and LOGON as the training data against each test corpus, including the in-domain corpus.

Results

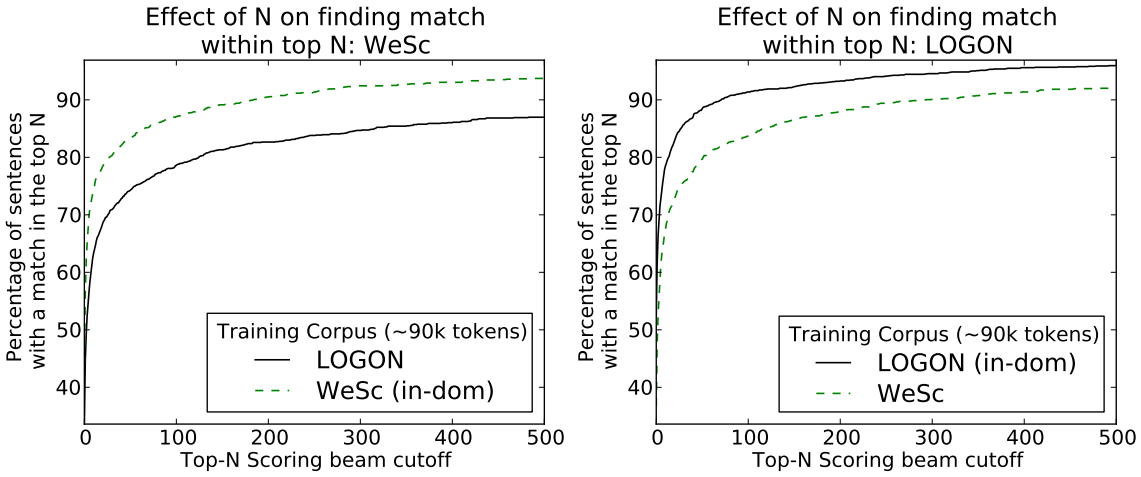
In Table 4.9 we give an indication of the size of cross-domain performance penalty using the ERG for these four domains, against the random baseline performance for reference (giving us an idea of how difficult the parse selection problem is for that corpus). It shows results using several metrics: exact match on the highest-ranked parse, exact match within the top 10, and precision, recall and F-score for EDM_{all} and EDM_{NA}.

Of primary interest here is how the performance over WESCIENCE data drops when the training data is purely LOGON versus purely in-domain WESCIENCE data, and vice versa. A related question is whether for a given new target domain (e.g. C&B or ROBOT1) we would expect each alternative training corpus to give equally useful accuracy figures. So, the results shown are over all test corpora using all WESCIENCE data or all LOGON data as training data.

We see here that in our two major test sets, a domain penalty can be seen across all evaluation metrics. For EDM_{NA}, which is the most comparable to dependency evaluations in other formalisms, the out-of-domain drop is 5–6%, which is in line with domain effects seen elsewhere. For the Acc_1 and Acc_{10} figures, there is a larger 10.5–15.8% drop, which is not unexpected for a stricter metric.

The results also show standard deviations, calculated by comparing results from 10 random sub-partitions of the test data. As we would expect, the standard deviation is larger for the exact match metric, where it ranges from 2.6 to 6.7%, than for the less-variable EDM F-score, where it is between 0.6 and 4.1%. We also see much greater variation for all metrics on the ROBOT1 corpus, and smaller standard deviation for exact match when the training corpus is from the same domain, although this does not hold for other metrics, and may not be meaningful. For the two test corpora where in-domain data is available, we see that the differences between accuracies from in-domain and out-of-domain training data is substantially larger than the standard deviations, but for the other two test corpora the differences caused by using the different training corpora are much smaller than the standard deviation for a single model, so the difference is too small to draw conclusions.

Comparing these results with the relative entropy figures in Figure 4.2, there is generally a correlation between a lower relative entropy and higher parse selection score using the corresponding model. The WESCIENCE and LOGON test corpora unsurprisingly show this effect most strongly, where there are the largest relative entropy differences against one training corpus versus the other (close to zero for the in-domain corpus). Both C&B and ROBOT1 generally show differences according to the metrics which agree with the differences in relative entropy against the training



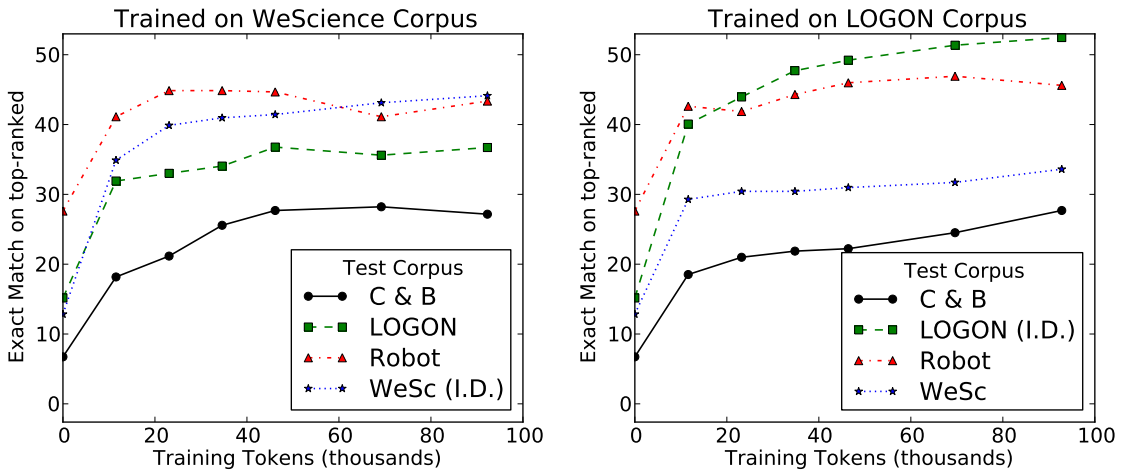
(a) WeScience (b) LOGON

Figure 4.4: Accuracy for gold-standard parse occurring within the top-N (i.e. Acc_N) against N

corpora (lower relative entropy corresponding to higher scores), but as we noted, these differences are not large enough to be clearly significant.

In Section 3.6.1, we argued that Acc_1 and Acc_{10} are easy-to-calculate representatives for a range of figures denoting treebanking utility. To illustrate this, in Figure 4.4 we show the Acc_N that would be obtained for values of N from one to 500 for some of the same in-domain and cross-domain training/test combinations that are shown in Table 4.9. The full graphs more directly show the effects of interest for treebanking, but they are expensive to create. Comparing between these graphs and Table 4.9, it appears that Acc_1 and Acc_{10} are representing some of this picture. In Figure 4.4(a), the two parse selection models for the WeScience data set result in a fairly consistent difference of around 11%, a consistency that can be seen just from the Acc_1 and Acc_{10} values in the table.[5] For Figure 4.4(b), the gap narrows as the beam widens, a fact again reflected by comparing Acc_1 and Acc_{10} for the relevant LOGON data set results. This narrowing gap shows that the impact of in- vs. out-of-domain training data is reduced when we are looking at more parses – probably because the less ambiguous sentences of LOGON are more likely to have less than N possible parses

[5]The WeSc treebank is parsed and treebanked against the top-500 parses according to a model trained on the WeSc data itself, so we might expect the accuracy to reach 100% in Figure 4.4(a), but the model we use here does not use the complete WeScience data set and has slightly different training parameters.



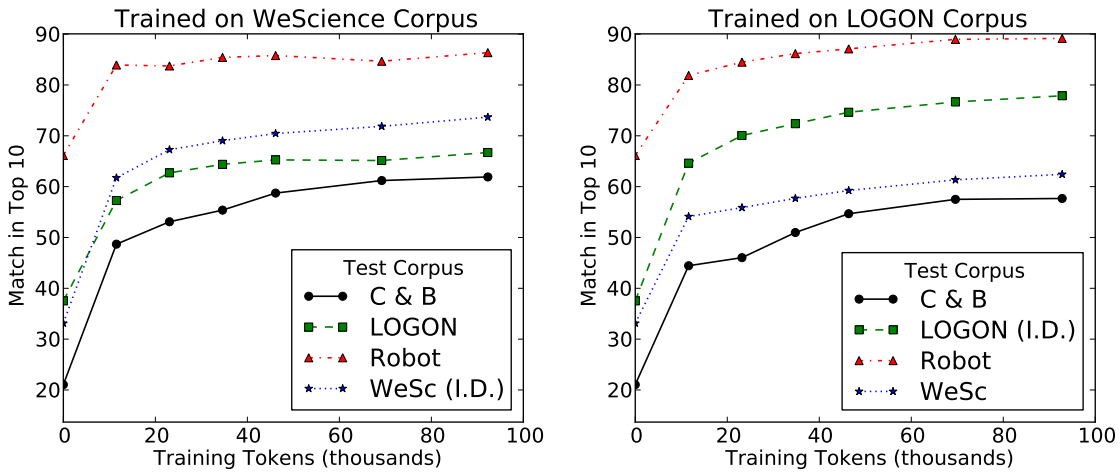
(a) WeSc (b) LOG

Figure 4.5: Learning Curves – Acc_1 (exact match). 'I.D.' denotes in-domain corpus

for higher values of N (i.e. all parses are within the beam), in which case the parse selection model does not matter.

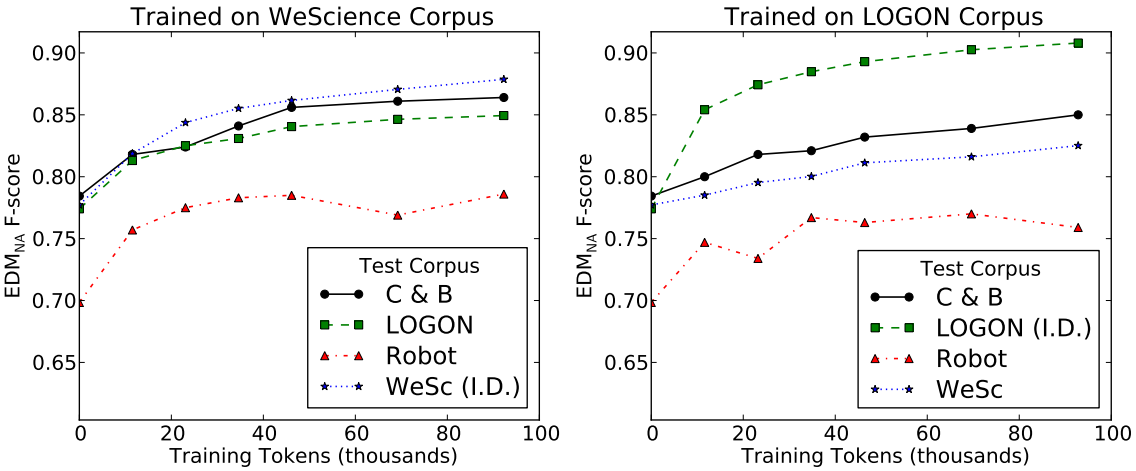
For the test sets for which we have no in-domain training data, there is little difference between the two parse selection models. They do, however, provide an interesting comparison between the exact match based metrics and the F-scores. In terms of EDM, the WeSc trained model gives almost the same results over C&B (within 2%) as it does for the in-domain test set; however, the Acc_1 results are much lower. For the robot1 data set, the EDM results are substantially lower than on any of the other test sets, but Acc_1 and Acc_{10} are high. To partially explain this, we can look back to Table 3.4. We saw there that the ambiguity level of the robot1 corpus, measured in parses per sentence, was much lower than that of the other data sets. This simplifies the parse selection task, since there are fewer analyses to consider. Conversely, the longer and more ambiguous sentences in the C&B corpus make it much more difficult to get every aspect of an analysis right. The relatively high F-scores for this corpus suggest that both parse selection models are doing a good job of returning a top parse with most of the dependencies and constituents correct. The different behaviours of the Acc_N and EDM metrics on different corpora show why we should consider both together in selecting the most appropriate parse selection model for a particular corpus. For the rest of this work, we primarily report results on exact match (Acc_1) and EDM_{NA}, with brief results for Acc_{10} where appropriate.

The results in Table 4.9 were produced by using all the available training data for each domain. We were also interested in how the results changed with the amount



(a) WeSc (b) LOG

Figure 4.6: Learning Curves – Acc_{10}. 'I.D.' denotes in-domain corpus



(a) WeSc (b) LOG

Figure 4.7: Learning Curves – EDM_{NA} F-score

of training data and so learning curves were produced for the same training and test sets. The learning curves obtained using the two different training corpora of approximately 8000 sentences each are shown in Figures 4.5, 4.6 and 4.7 using the

exact match metric as well as EDM F-score over four different test-domains. In each case, the in-domain corpus is marked as such.

The basic shape of the curves is unsurprising. Generally, the curves are monotonically increasing, so more training data of any type produces better results. This effect continues even when relatively large amounts of training data were already being used, but as we would expect, there is some flattening off in these curves, as the more training data we have, the less incrementally valuable it is. This levelling off is more noticeable when the training data is entirely out-of-domain – suggesting that there is a limit to the amount of out-of-domain data which can usefully improve parsing performance, at least in isolation (i.e. when not in combination with in-domain data, which is discussed in more detail below). Indeed, it is possible that too much (solely) out-of-domain data could have a detrimental effect. There are small drops in some of the learning curves as out-of-domain data is added, particularly in the Acc_1 evaluation, for C&B, ROBOT1 and LOGON at various points, although this is at most a very minor effect.

Again, we can clearly see that Acc_1 and EDM give a different picture of performance on the C&B and ROBOT1 corpora. Comparing these figures tells us something else about the different metrics: in some situations, Acc_1 may be more useful in differentiating the success of a given model on multiple domains. The exact match metric shows more noticeable relative changes than EDM when we make subtle changes in the model and the domain, emphasising the importance of domain for treebanking or other applications where we demand an exact tree match. The EDM results cluster more closely in absolute terms for most data sets regardless of the amount of training data or the domain, but there are still reliable, albeit small, changes as the amount of training data is altered. This follows from the more 'forgiving' nature of the EDM-based evaluation, but it also tells us something about the grammar: given a very small amount of training data from any domain, the top-ranked parse will have most of the dependencies correct.

For maximum sensitivity in parse selection experiments (as well as tractability in experiments with many successive runs such as grid searches), we would argue that the exact match metric is undoubtedly useful, and provides a complementary perspective to EDM.

If the EDM metric, as intended, more closely reflects the performance we could expect in downstream applications, it may appear that these are more robust to changes in domain. However, it is possible that for these applications using the parser output, it is the error rate which is more important. From this perspective it seems EDM can be more sensitive to choice of training domain than exact match. From Table 4.9, we can see that over WESCIENCE, for Acc_1, the error rate goes from 66.4% to 55.9%, a 16% relative reduction, when moving from out-of-domain to in-domain training data, while the relative reduction in EDM F-score error rate (from 17.5% to 12.1%) is 31%. Similarly for LOGON by using in-domain data, we get a 25% relative error rate reduction for Acc_1, and 39% for EDM.

It is also instructive to compare error rate reduction relative to the random baseline (although it is not truly random as it incorporates the top-500 parses according to a model which has training data from WeScience and LOGON). For EDM the relative reduction from using out-of-domain data is 21% for WeScience, and 33% for LOGON. This is smaller than the reduction in error rate when we move from out-of-domain to in-domain data (31% and 39% respectively, as quoted above), suggesting that a tuned parse-selection model is quite important – it can give more of a performance boost over an informed but unmatched training corpus than that unmatched corpus does over a random selection. However, further experimentation would be required to determine whether the error rate reduction is more meaningful in terms of downstream utility.

Additionally, it seems that not all corpora are equal in terms of cross-domain applicability. From Figures 4.5 and 4.7, we can see that WeScience as the only training corpus gives slightly better results for C&B (apart from the slightly higher Acc_1 from the full 93,000 token training sets when using LOGON), as we would predict from the relative entropy figures. On the other hand, the best training corpus for robot1 is less obvious. Indeed, it seems that training data beyond the first 11,000 tokens does very little, and sometimes decreases performance. LOGON gives slightly higher exact match performance, although the figures are so variable that the differences may not be meaningful. In general, training data does little for robot1 probably due to the very different nature of the corpus compared to the data we have available, with the smallest absolute improvements and error rate reductions over the random baseline of any of the test corpora.

4.4.3 Models from Unmodified Concatenated Data: Concat

Next we looked at some simple strategies for alleviating the cross-domain penalty. From the work described in Section 2.6, there are various strategies for achieving this. Domain-adapted POS-taggers or supertaggers (Lease and Charniak 2005; Rimell and Clark 2009) have been successful in the biomedical domain, in the latter case for helping to constrain the parser search space. This is not something we investigate here since domain-adapted tagging models are not readily available, and at least in the parsing configuration we use, POS tagging is only used for unknown words, but it is a possible area for future work. Given that we already have relatively large training corpora in separate domains, a logical first step is to see how much manually-annotated data we need in order to improve accuracy in a new domain, whether we should combine this with existing data, and how best to do so.

First, we evaluated a relatively naive method for combining training data from the LOGON and WeScience domains into one model. In this strategy, denoted Concat, we simply concatenate the training data sets and train a MaxEnt model from this, in the same way as the "combined" method of Hara *et al.* (2005). To

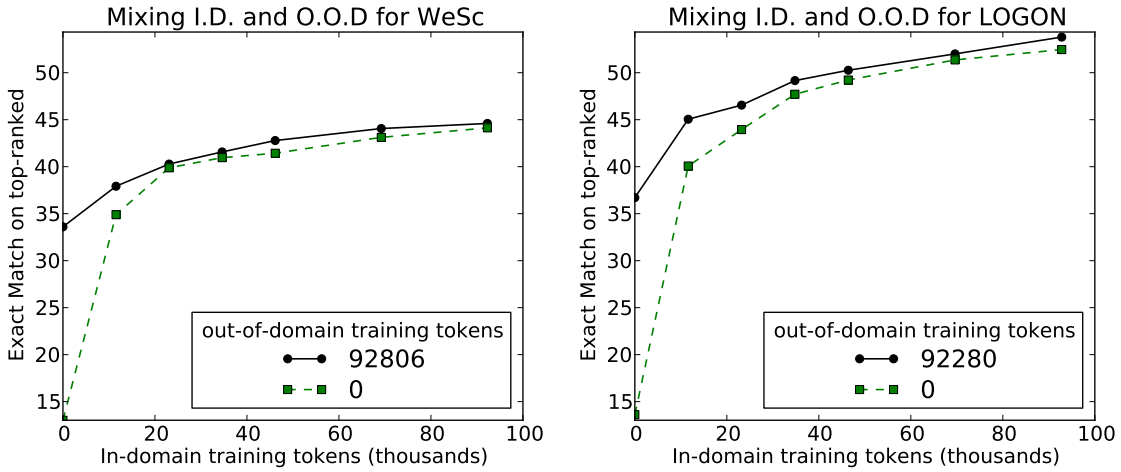
simulate the effects of differing amounts of treebanking effort, we varied the amount of in-domain training data used in each model.

Results

Having measured how much our performance is affected by using only out-of-domain data, we now look at the results from the simple concatenation of the two corpora of training data (CONCAT). There are likely to be two significant factors here — the amount of in-domain data, and the amount of out-of-domain data.

One common scenario might be that we have a fixed volume of training data, and wish to know how much in-domain data we need to treebank to substantially improve performance when combining it using CONCAT with the out-of-domain data, or alternatively how much improvement we can expect from some volume of in-domain data. Secondarily, it is interesting to investigate whether it is possible to add too much out-of-domain data compared to in-domain — is there ever a situation where more data does not improve performance?

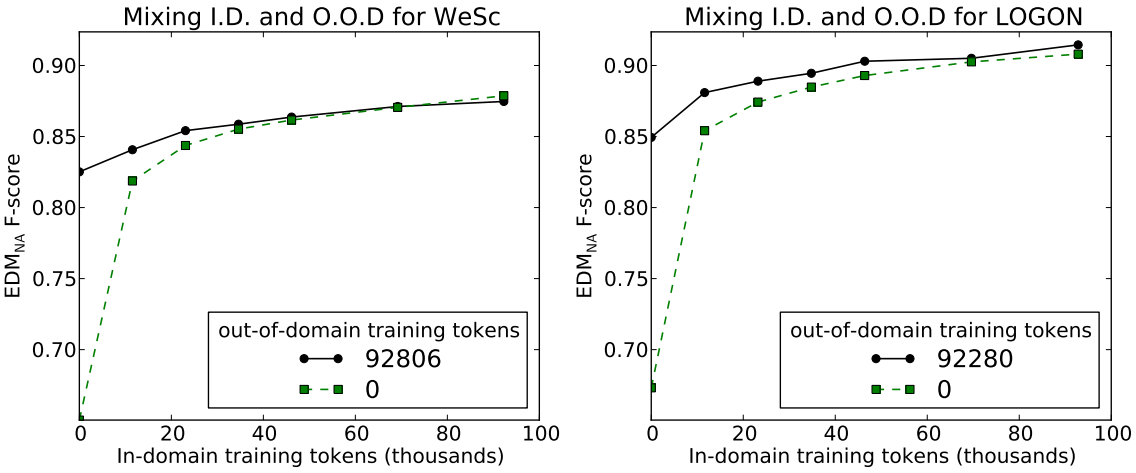
In Figures 4.8 and 4.9 we show some indicative results for these questions, using no out-of-domain data or all of it, and evaluate how they interact with different-sized in-domain training corpora.



(a) WESCIENCE (b) LOGON

Figure 4.8: Combining in-domain and out-of-domain training data using CONCAT: training a model from concatenated training corpora: Exact match

One interesting result from these figures is the effect that even a small amount of in-domain training data can have. Using a model built only on approximately 11,000 tokens on in-domain data, we get better results than the large out-of-domain trained model. Over the WESCIENCE corpus, once we have around 23,000 tokens of



(a) WeScience (b) LOGON

Figure 4.9: Combining in-domain and out-of-domain training data using Concat: training a model from concatenated training corpora: EDM

in-domain data, the out-of-domain data is having almost no effect. There does not appear to be a negative effect from including the out-of-domain data, but the benefit is almost non-existent for WeScience and very slight for the LOGON corpus. We also see that the additional benefit of adding more in-domain data tails off once we have a reasonable quantity (i.e. a few thousand sentences), which is a similar effect to the learning curves.

4.4.4 Linear Combinations of Trained Models: Combin

An alternate approach to Concat, which we denote Combin, is to use MaxEnt models derived from simple arithmetic combinations of other pre-trained models, similar to McClosky *et al.* (2010). As we described in Section 2.3.1, a MaxEnt model is simply a set of feature weights (the λ_i values from (2.31)). If we have two trained MaxEnt models A and B, and we wish to use weighting parameters α and β to combine these models into a new model C, the weight λ for a given feature in C will be given by $\lambda_C = \alpha\lambda_A + \beta\lambda_B$ where λ_A and λ_B have values of zero if the given feature is not explicitly encoded in the model. Since parse ranking uses the unnormalized MaxEnt score, only the ratio $\frac{\alpha}{\beta}$ matters, so we constrain α and β to sum to one, preventing multiple equivalent models. For example, assume we have the following weighting parameters and feature/weight pairs (i.e. vectors of λ values) for each model:

$$\alpha = 0.3 \qquad \beta = 0.7$$
$$m_A = \{1 : 1.5, 2 : 2.0\}$$
$$m_B = \{1 : 0.5, 3 : -1.0\}$$

The resulting model is simply the weighted linear combination:

$$m_C = \{1 : 0.8, 2 : 0.6, 3 : -0.7\}$$

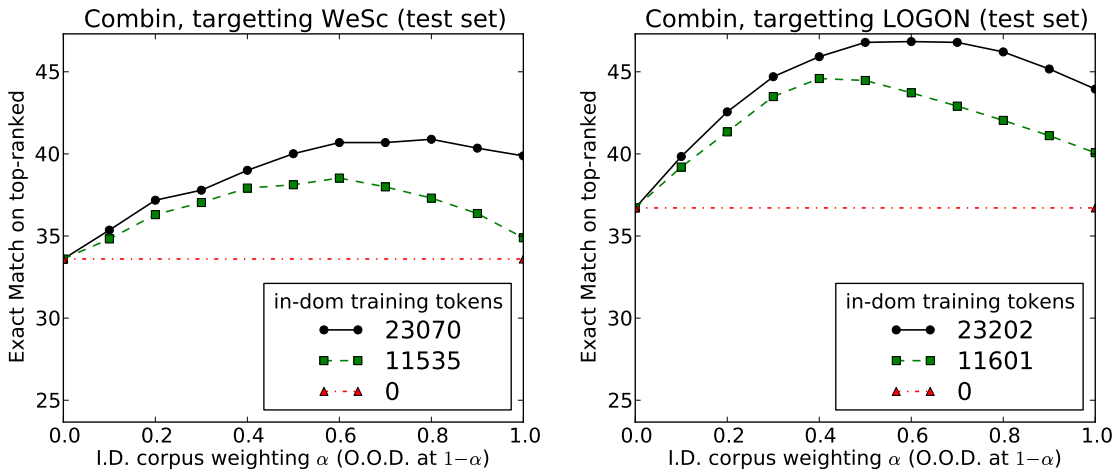
For our experiments here, the pre-trained models come from exactly one single-corpus treebank before they are arithmetically combined.

This is a potentially useful strategy as it allows flexible and fast tuning of parsing models with the possibility of improving performance in a number of cases. Using this strategy, we might as a starting point create a combined model by setting the weighting of each corpus in proportion to the number of sentences in the treebank from which the model was trained. We may expect this to produce somewhat similar results to the aforementioned strategy of training a model from the concatenated treebanks, but there is no guarantee that this is the case.

However, often we might expect better results by biasing this weighting somewhat — in general, we probably wish to give more weighting to the model trained on a more similar treebank. It is clear that this could be useful in the situation where we have a small treebank and trained model in a new domain that we wish to use most effectively, alongside an existing model from a larger treebank – by modifying the weightings, and probably boosting the weighting for the corpus in the new domain, we have a way to make maximal use of this small amount of training data, without needing to discard anything from the larger established treebank. A related situation is where we have two models trained from treebanks in two domains, and wish to parse a third domain for which we have no treebanked data. Intuitively, we should make use of all the data we have, but it may improve performance if we give a higher weighting to the domain that is more "similar" to the target domain — i.e. providing a higher accuracy parse selection model. McClosky *et al.* (2010) shows some *a priori* ways to determine this for a different parsing framework. We do not investigate this in detail here, but as described below, we do examine some techniques to optimise the parameters when we know that the target domain very closely matches some small training corpus.

Results

The graphs in Figures 4.10 and 4.11 show results for two different sized in-domain models (with the fully out-of-domain model shown as a baseline), and how the performance varies as the weighting between the in- and out-of-domain models is varied.



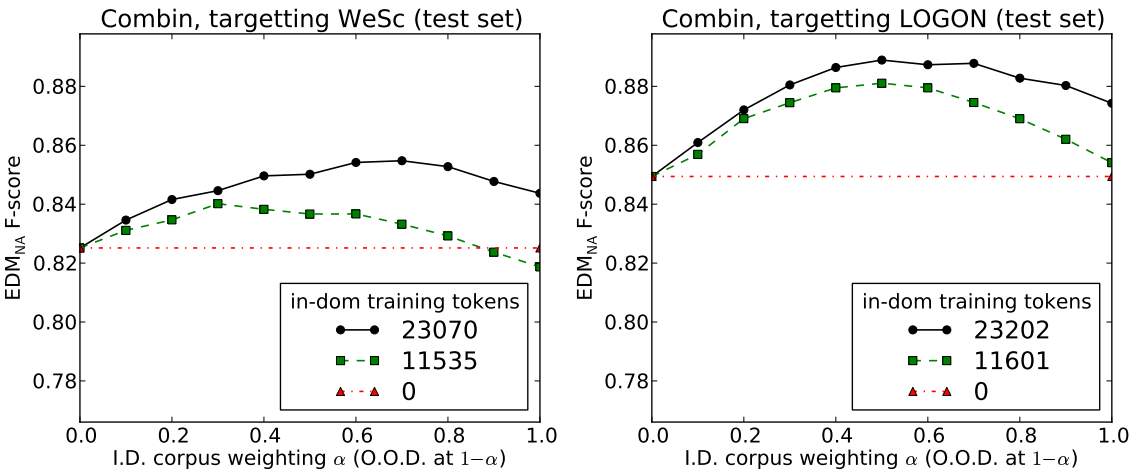
(a) WeScience (b) LOGON

Figure 4.10: Exact match scores for Combin: linearly interpolating between two models

Clearly, the choice of weighting makes a relatively large difference: the worst performing weight combination is substantially worse than simply using the Concat strategy. All curves show a similar parabolic shape, with the optimal mixture point being further to the right for the larger in-domain models in each case, meaning that it is better to weight the in-domain model even more heavily when it is larger, presumably because it is more reliable (compared to a model from a smaller treebank) as well as being closely matched to the domain.

4.4.5 Monolithic Models from Duplicated Data: Duplic

We also investigate another method for weighting two training corpora differently when they have different sizes and probably different levels of appropriateness for the test data. In strategy Duplic, we simply duplicate one of the corpora some number of times, as if the training corpus consisted of multiple copies of every sentence, then concatenate the data to the other corpus and extract training features and train a model in the same way. As noted in Section 4.4.1 we pay attention to certain 'counts' associated with each maximum entropy feature – in particular the relevance count threshold, for how many times a feature has been used to distinguish between good and bad parses. We take care to make sure these counts are appropriately incremented in the duplicate corpora as if the sentences are genuinely distinct. This is obviously a



(a) WeScience (b) LOGON

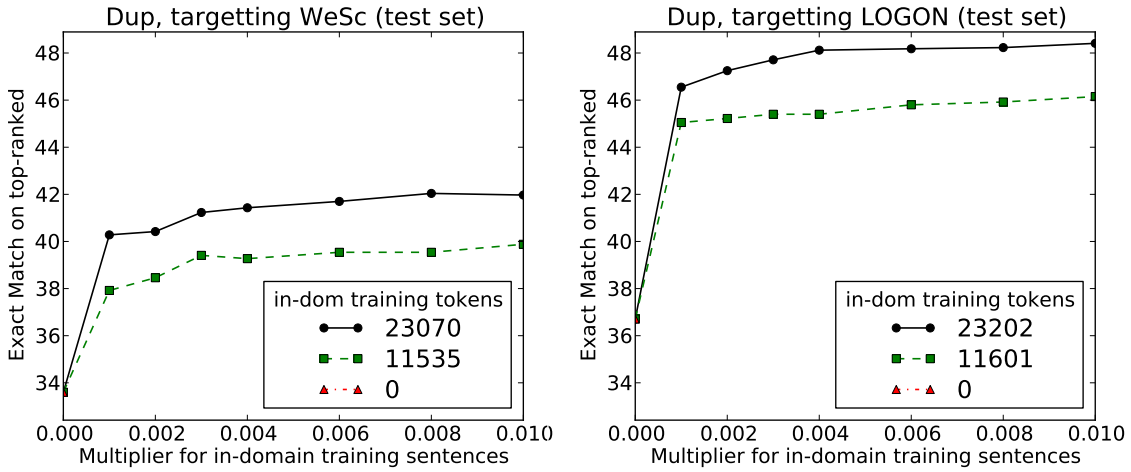
Figure 4.11: EDM$_{NA}$ F-scores for Combin: linearly interpolating between two models

generalisation of Concat, but we treat it separately as Concat is the usual default approach for combining such corpora.

In comparison to Combin, this is a more expensive approach to optimise over different corpus combination parameters, since we need to retrain the maximum entropy model (and, currently, recreate the feature cache) for each different weighting of the corpora, rather than building at most two models and performing a rapid linear combination. Nonetheless, this might lead to a more accurate model than Combin, justifying the extra cost, although in a system which is trying to optimise between several different corpora (rather than just two as we are doing here), this extra cost may be prohibitive. If the parameters are known in advance however, and we are merely trying to build a model, the difference in training time between the two approaches is minimal.

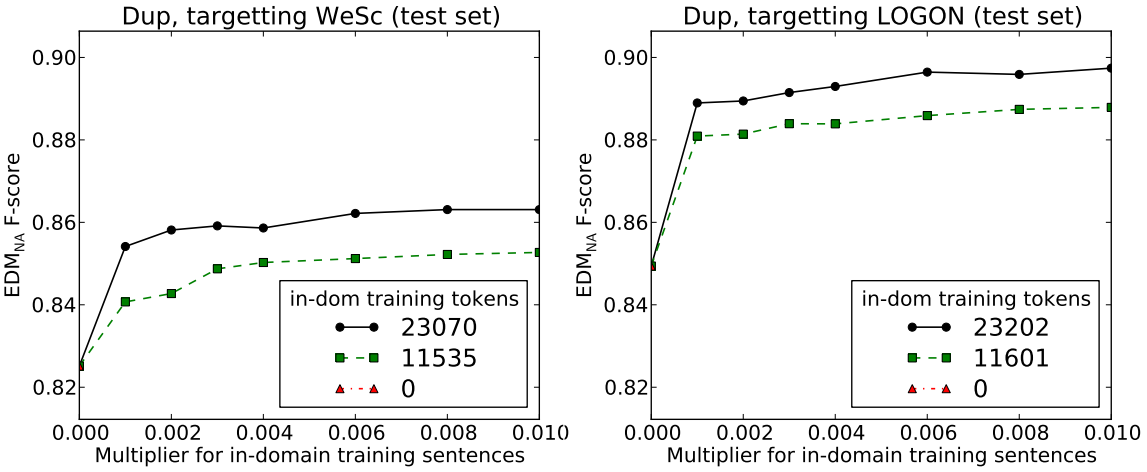
Results

Figures 4.12 and 4.13 show the results for Duplic, which duplicates the smaller in-domain corpus some integral number of times and combines it with the larger out-of-domain corpus before training a model. Again, we show graphs for two different sized in-domain training sets, this time varying how many times the in-domain set was duplicated before training the model. This is better than Combin at improving performance (excluding the in-domain corpus at zero weighting factor which we include for completeness), as well as more robust for parameter selection. The exact



(a) WeScience (b) LOGON

Figure 4.12: Exact match scores for Duplic: duplicating the in-domain data set multiple times



(a) WeScience (b) LOGON

Figure 4.13: EDM$_{NA}$ F-scores for Duplic: duplicating the in-domain data set multiple times

match accuracy over the full corpus usually increases monotonically, so it is almost always of some benefit to weight the smaller in-domain corpus more heavily (however, presumably at some point these benefits vanish). The same is true generally for EDM, although the increase is less convincing in this case.

4.4.6 Optimising Combination Parameters

The graphs in Sections 4.4.3 and 4.4.5 allowed us to see the optimal parameters for these particular models and test sets, and indicated that some of these best-performing parameters can give superior results to the naive CONCAT method in the best case. However, this does not address the question of determining the optimal set of parameters for COMBIN or DUPLIC without evaluating over the test data (which would not be realistic for a real-world application where the gold-standard annotations presumably do not exist). In the same way that we can tune the grid-search parameters using cross-validation over the training data, we can also use a similar approach to optimise the weights or multiplication factors for each domain. Specifically, we can use our small in-domain training set, divide that corpus into n cross-validation folds, and combine the training data from each fold with the complete data-set from the larger out-of-domain corpus using COMBIN or DUPLIC with various sets of parameters, then test over the test fold.

For 8-fold cross-validation, which we use here, this means we must train and test 8 models per set of parameters. For 7 different DUPLIC and 10 different COMBIN parameters, this means 136 different models are required per test-run. As in the grid-search discussed above, we rerank rather than reparse these sentences.

By aggregating these test fold results, we can select the optimal parameters. For tractability, we only calculate Acc_1 (using reranking of the parse forest) for the cross-validation results, and simply select the highest accuracy combination as 'optimal'. A more sophisticated approach could also take into account factors such as consistency, as measured by a low variance across the cross-validation folds, and also pay attention to the other scoring metrics.

Of course, we cannot guarantee that the parameters will indeed be optimal over the test data. For this reason we evaluate the various parameter combinations both using cross-validation and over the test data, giving us an indication of how close we are to the optimal parameter values for unseen data by selecting the best performer in cross-validation.

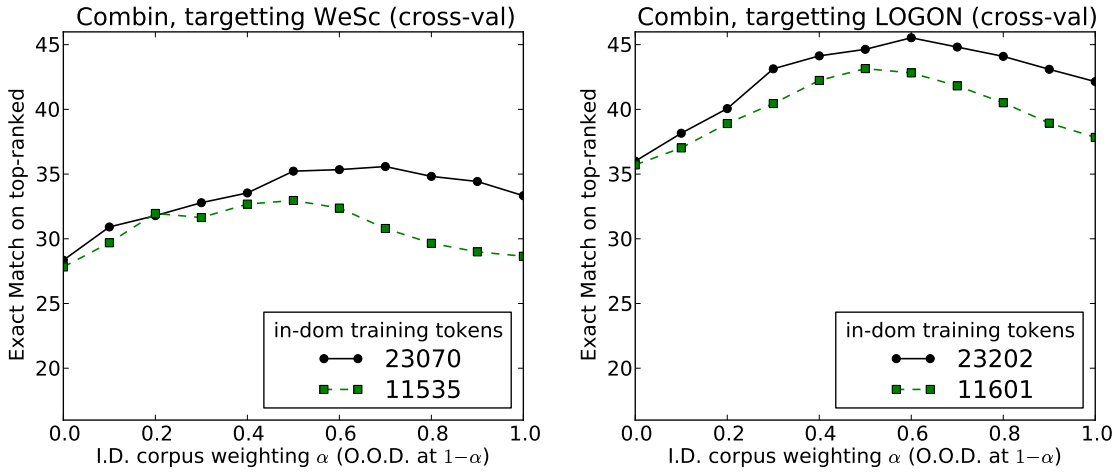
Results

We show the cross-validation results for exploring the same parameter space as in Sections 4.4.3 and 4.4.5 in Figures 4.14 and 4.15. Figure 4.14 is the analogue of Figure 4.10 but using cross-validation over the training set. Comparing these two, it

seems likely that cross-validation provides a reasonable estimate of performance over the development set for COMBIN, although the results are slightly noisier.

Figure 4.15 is the cross-validation analogue of Figure 4.12. In some cases cross-validation provides a reasonable estimate of test set performance although it is not particularly helpful over the smaller set of WESCIENCE training data, which performs best in cross-validation with a $(2, 1)$ weighting for in-domain against out-of-domain, while over the test set, a $(10, 1)$ weighting is substantially better, both for EDM and exact match.

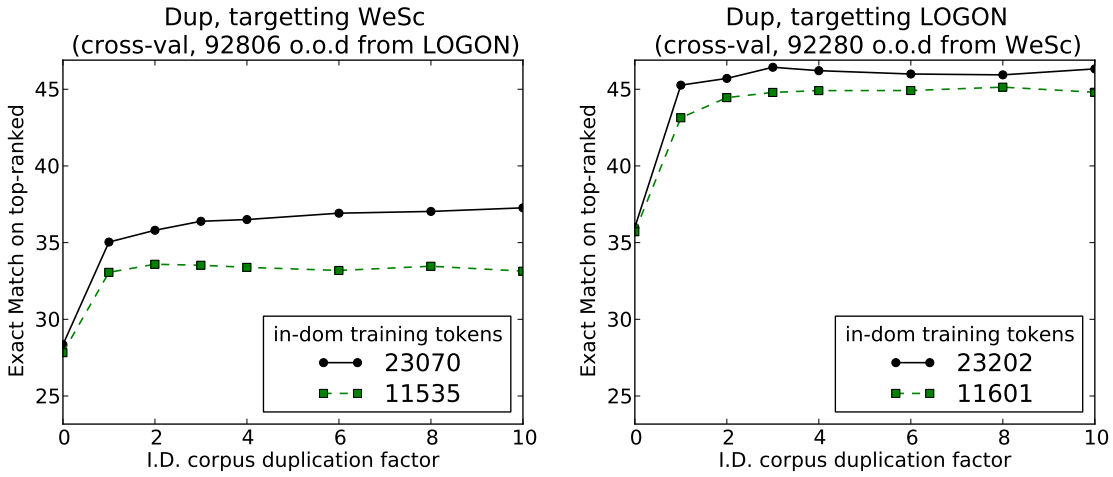
While the cross-validation results seem to provide a reasonable estimator of performance over unseen test data, we can evaluate this more rigorously by measuring the Pearson correlation between the cross-validation results for Acc_1 and the results using the same parameters on the held-out test data for both Acc_1 and EDM. This figure is usually 0.99 or more for DUPLIC, and 0.93 or more for COMBIN. The only exceptions are for WESCIENCE with 769 sentences of training data, which using DUPLIC gives 0.95 for Acc_1 and 0.88 for EDM, and using COMBIN gives 0.80 and 0.88. In general, the cross-validation estimates are fairly predictive of the results we can expect over held-out data in the same domain, particularly for DUPLIC, meaning that selecting the best parameter combination from cross-validation is a potentially useful strategy, at least in terms of predicting relative changes in accuracy over unseen data.



(a) WESCIENCE (b) LOGON

Figure 4.14: Aggregated accuracy using COMBIN over the eight cross-validation folds of the in-domain corpus, where the in-domain training data is the other seven folds and the out-of-domain data is the entire other corpus

For handling real-world data with an unbalanced in-domain corpus, we might generally pick the best-performing parameter set from cross-validation and apply that parameter combination to new data. While the correlation suggests that such a



(a) WeScience (b) LOGON

Figure 4.15: Aggregated accuracy using Duplic over the eight cross-validation folds of the in-domain corpus, where the in-domain training data is the other seven folds and the out-of-domain data is the entire other corpus

parameter should reliably predict relative changes using the same parameters over unseen data, this does not guarantee an absolute significant performance improvement. We applied this technique to the corpora used above. These results can be read from the previous graphs for WeScience and LOGON, but they are presented in a more convenient form in Table 4.10, which shows the auto-selected best-performing parameters from cross-validation for both combination techniques, and the results that we can obtain using these over unseen test data. For another point of comparison, we also applied the same technique using cross-validation over the robot1 development set (which has not yet been used until this point) combined with different combinations of the WeScience and LOGON training sets and tested using the same test data as used in previous sections.

We calculate statistical significance using the "compute-intensive" shuffling procedure of Yeh (2000). In a given iteration, for all results which differ between the new method and the baseline, we swap each result pair with probability 0.5 and test whether the new synthesised results differ by more than was observed between the actual results of the new method and the baseline (which would suggest the difference is due to chance and the null hypothesis could be true), incrementing count c if this is the case. Repeating for some large number of iterations t, the p-value can be estimated to be at most $\frac{c+1}{t+1}$. After calculating this, following Cahill *et al.* (2008:105), we

Test	Train Tokens I.D.	Train Tokens O.O.D	Meth	Weights	Acc_1	Acc_{10}	EDM_{NA}
LOG	WeSc:11.6k	WeSc:92.3k	baseline		45.0	72.4	0.881
			Duplic	(8, 1)	45.9*	73.5*	0.888***
			Combin	(0.5, 0.5)	44.5	71.7	0.881
LOG	WeSc:23.2k	WeSc:92.3k	baseline		46.5	73.9	0.889
			Duplic	(3, 1)	47.7**	74.6	0.892
			Combin	(0.6, 0.4)	46.8	73.9	0.888
robot1	robot1:4.5k	LOG:92.8k	baseline		74.0	93.5	0.925
			Duplic	(3, 1)	75.1*	93.6	0.924
			Combin	(0.5, 0.5)	76.1	92.7	0.916
robot1	robot1:4.5k	WeSc:92.3k	baseline		75.1	93.3	0.922
			Duplic	(4, 1)	75.5	93.3	0.929
			Combin	(0.7, 0.3)	76.3	91.2	0.910
robot1	robot1:4.5k	WS+LG:92.5k	baseline		75.5	93.1	0.925
			Duplic	(10, 1)	77.8**	93.5***	0.936
			Combin	(0.5, 0.5)	77.2	92.5	0.926
WeSc	WeSc:11.5k	LOG:92.8k	baseline		37.9	66.9	0.841
			Duplic	(2, 1)	38.5	67.8*	0.842
			Combin	(0.5, 0.5)	38.1	66.1	0.836
WeSc	WeSc:23.1k	LOG:92.8k	baseline		40.3	69.0	0.854
			Duplic	(10, 1)	42.0**	71.3***	0.863***
			Combin	(0.7, 0.3)	40.7	69.5	0.854

Table 4.10: Comparison of best results for Combin and Duplic, selected by cross-validation over training data, and evaluated over standard test sets used previously. Asterisks denote statistical significance of improvements over Concat baseline: * for $p < 0.05$, ** for $p < 0.01$ and *** for $p < 0.001$ (corrected for 7 runs)

correct for the multiple training/test corpus combinations (seven in this case), using the p-value correction technique of (Cohen 1995):

(4.4)
$$\alpha_e \approx 1 - (1 - \alpha_c)^m$$

for m pairwise comparisons, where α_c is the per-comparison error and α_e is the per-experiment error. For seven comparisons, this means that to obtain a corrected p-value of 0.01, the per-comparison value must be less than 0.0014.

As reported in Table 4.10, most of the parameters that were selected as a result of the cross-validation procedure for DUPLIC produce statistically significant improvements over the baseline on unseen data for at least one metric, and often across all three — even though we know from the test data that some of the parameters are sub-optimal, and oracle parameter selection could produce superior results. COMBIN is less promising – indeed, the value selected can be worse than the baseline. But if we restrict ourselves to DUPLIC, it seems that this simplistic and imperfect cross-validation technique to tune parameters can produce statistically significant improvements in accuracy and F-score at the cost of CPU-cycles, with no observed performance drops against the baseline. For 23,000 tokens of WESCIENCE, we get a relative reduction in the exact match error rate of 2.9% and 6.1% in the EDM F-score error rate, almost as much as we would get from treebanking an extra 23000 tokens. Additionally, on the basis of the test data we have looked at here, it seems that a weighting around $(8, 1)$ would be a robust performer, so the cross-validation step may not be necessary — although it may not be as applicable for all data.

All of this is of course predicated on being able to closely match the test and training corpora – we must be sure they are from the same domain, which may not be as easy in practice as it is with curated data sets. The 'shuffling' we used to divide up the corpora may mean that the match between the different corpus sections is closer than we would see in real-world data, so this could have slightly inflated the performance improvements from upweighting the in-domain data. The relative entropy comparison we discussed in Section 4.2.2 suggests one possible strategy for this, and although the comparison of lexical rules depended on hand-annotated gold standard data, it is possible the best parse from an automatically-created parse tree would be a reasonable substitute. Relatedly, we might want to apply a more informed approach along the lines of McClosky *et al.* (2010), which suggests a model on the basis of how closely a test corpus is predicted to match each possible training corpus.

4.4.7 Self-training

As noted in Section 2.6.1, research on domain adaptation in at least two other formalisms (McClosky and Charniak 2008; Honnibal *et al.* 2009) has found that a successful domain adaptation strategy is self-training. In essence, this is running the parser over some unannotated corpus, and treating the highest-scored candidate parse

Test	Train Tokens			Acc_1	Acc_{10}	EDM_{NA}
	ST Parsed	ST Model	Gold Data			P / R / F
LG	–	–	WS:92.3k	36.7	66.7	0.862 / 0.837 / 0.849
	LG:95.7k	WS:92.3k	–	35.9	63.9	0.863 / 0.857 / 0.860**
	LG:95.7k	WS:92.3k	WS:92.3k	36.7	64.2	0.866 / 0.860 / 0.863***
WS	–	–	LG:92.8k	33.6	62.4	0.842 / 0.808 / 0.825
	WS:103.4k	LG:92.8k	–	33.7	60.2	0.843 / 0.836 / 0.839***
	WS:103.4k	LG:92.8k	LG:92.8k	34.0	60.8	0.844 / 0.837 / 0.840***

Table 4.11: Self-training experiments, simulating parsing with no gold-standard in-domain data available. The first row for each test corpus using only gold out-of-domain training data is the baseline (results repeated from Table 4.9). The subsequent rows show the result of parsing an in-domain corpus ('ST Parsed') with a model trained on approximately 90k tokens of out-of-domain data ('ST Model'), and then treating the top-ranked parse for each sentence as pseudo-gold. Where shown ('Gold Data'), the same gold data used to train the baseline is also used when training the new parse selection model, augmenting the self-trained pseudo-gold data (using CONCAT). ** and *** indicates results which are statistically-significant improvements over the baseline (**: $p < 0.01$; ***: $p < 0.001$) corrected for four runs.

as "pseudo-gold", then using this as training data for some component in the parsing pipeline.

In both McClosky and Charniak (building on work from McClosky *et al.* (2006a)) and Honnibal *et al.*, output from one component in the pipeline is used to train another. Our work here differs in that PET, the parser we are using, has a single-step parsing process, where the parse forest is unpacked at the same time as the trees are scored according to the model, in descending order of parse score. There is a POS-tagging phase to handle unknown words, but this is far less closely tied to the parsing procedure than the supertagging is for the C&C parser. So while this other work may suggest that we can expect some utility from self-training, it is not at all clear that this should work when we have only one component to provide training data; indeed, as mentioned previously, McClosky *et al.* found that it could have a negative impact.

In terms of the self-training architecture, our work here is more naturally parallel to the *simple self-training* strategy (Sagae 2010) discussed in Section 2.6.1, although there are differences as well — most pertinently, we are using a discriminative model rather than the generative PCFG of the Charniak (2000) parser.

Additionally, there are other reasons to suspect that a simple self-training procedure may be useful here. In particular, the fact that we are working with a constrained precision grammar affects the search space beyond how it is affected by the parse selection model, so it is possible that a discriminative learner can discern useful information from an automatically-ranked parse forest which embodies some of these constraints.

Following this related work and the theme we have established in this chapter, we examine self-training as a tool for domain-adaptation (rather than, for example, completely unsupervised parse selection). This could be applicable when we have no in-domain data, which usually results in the performance penalty discussed in Section 4.4.2, or when we have only a small amount of in-domain data, as we investigated in Sections 4.4.3, 4.4.4 and 4.4.5.

The self-training strategy we test is to parse some set of sentences using PET with the best parse selection model we have available for the target domain – either a purely out-of-domain model, or a partially adapted model. We then mark the top-ranked tree from each forest as pseudo-gold, and use all other trees in the top-500 as negative training instances (which is another point of differentiation from the work cited above, which use only the top-ranked parse for positive training instances). From this, we train a parse selection model in the usual way, generally after combining the data using CONCAT with the same training data that was used to create the first-iteration parse selection model.

We use only the sentences which are within our regular gold training sets to create pseudo-gold data. There is no particular reason why this needs to be the case, since we would only need to parse some unseen sentences from the same domain to create more data, but it does enable easy comparison with using gold-annotated data. Additionally, this keeps the number of sentences manageable for training the parse-selection model with TADM (even with this constraint, this process can use up to 20GB of memory for some configurations described below). This means we have at most 9,000 sentences of self-training data (unlike when we train from hand-annotated data, we include those sentences which were rejected in the treebanking phase, to simulate using real-world data). This also differs from McClosky and Charniak and Honnibal *et al.*, who use 270,000 or 185,000 training sentences respectively.

This is not intended as an exhaustive exploration of the possibilities of self-training with this grammar; rather, it should be viewed as an initial investigation of the possibilities as part of a domain-adaptation strategy, to augment what we have already explored in this chapter.

Results

The first set of experiments simulates having only out-of-domain human-annotated data, and is summarised in Table 4.11. For testing the impact of self-training, we evaluate using only self-training data to create the model, as well using self-training

Test	ST Parsed	Gold Training I.D.	Gold Training O.O.D.	Gold Training Weight	Acc_1	Acc_{10}	EDM_{NA} F
LG	–	LG:11.6k	WS:92.3k	(1, 1)	45.0	72.4	0.881
LG	–	LG:11.6k	WS:92.3k	(8, 1)	45.9	73.5	0.888
LG	LG:83.7k	LG:11.6k	WS:92.3k	(8, 1)	45.8†††	71.6	0.899***†††
LG	–	LG:23.2k	WS:92.3k	(1, 1)	46.5	73.9	0.890
LG	–	LG:23.2k	WS:92.3k	(3, 1)	47.7	74.6	0.892
LG	LG:71.7k	LG:23.2k	WS:92.3k	(3, 1)	47.6*†††	73.6	0.906***†††
WS	–	WS:11.5k	LG:92.8k	(1, 1)	37.9	66.9	0.841
WS	–	WS:11.5k	LG:92.8k	(2, 1)	38.5	67.8	0.842
WS	WS:90.4k	WS:11.5k	LG:92.8k	(2, 1)	39.1*†††	65.6	0.857***†††
WS	–	WS:23.1k	LG:92.8k	(1, 1)	40.3	69.0	0.854
WS	–	WS:23.1k	LG:92.8k	(10, 1)	42.0	71.3	0.863
WS	WS:77.5k	WS:23.1k	LG:92.8k	(10, 1)	42.3**†††	69.0	0.873***†

Table 4.12: Self-training experiments, simulating parsing with limited gold-standard in-domain data available. The first row for each test corpus using only gold training data is the CONCAT baseline, and the second is the results for DUPLIC with parameters selected through cross-validation (both repeated from Table 4.10). The subsequent rows show the result of self-training – parsing the remainder of the in-domain corpus ('ST Parsed') for which we have no gold data with the same DUPLIC model, and then treating the top-ranked parse for each sentence as pseudo-gold. For self-training, the gold data used to train the first iteration parse selection model (i.e. the best DUPLIC model) is also used when training the new parse selection model, augmenting the self-trained pseudo-gold data. *, ** and *** indicate statistical significance over the CONCAT baseline (*: $p < 0.05$, **: $p < 0.01$, ***: $p < 0.001$). †, †† and ††† indicate statistical significance over the DUPLIC result without self-training (†: $p < 0.05$, ††: $p < 0.01$, †††: $p < 0.001$), all corrected for 8 runs.

data combined with out-of-domain gold data. The self-training data itself is the complete in-domain corpus apart from the test sections, parsed using the out-of-domain model to select the pseudo-gold tree. For the baseline, we reuse a subset of the results from Table 4.9 — namely, those for LOGON and WESCIENCE using a parse selection model created from only the out-of-domain training data.

Looking at the EDM scores, the results are promising. With no manual annotation, we obtain a significant boost in F-score of 1.5%–1.8% (or slightly less for the variants which exclude the explicit gold data). Comparing this to Figure 4.9, we

can see this is roughly equivalent to adding 7,000–11,000 tokens of human-annotated data, which corresponds to a small treebank of around 600–800 sentences.

On the other hand, the exact match scores are equivocal at best. The biggest improvement in Acc_1 of 0.4% is not statistically significant, and in some cases it decreases, although not when we use the gold data. Additionally, all self-training configurations decreases Acc_{10} somewhat. The opposing effects we see on these different metrics are discussed in more detail below.

Given that at least some metrics show a benefit from using self-training, it is natural to investigate whether this is useful to enhance performance when we have limited in-domain data. Specifically, the best method we have seen so far for making maximal use of limited in-domain data is to use the optimisation techniques of Section 4.4.6 to tune parameters for the DUPLIC combination method from Section 4.4.5. An obvious question is whether we can use self-training to improve performance beyond the maximum we were able to achieve for the particular sets of unbalanced training corpora we have seen.

In Table 4.12, we show results for the CONCAT baseline as well as those for DUPLIC with the weighting selected through cross-validation. For the self-training results, we used the corresponding best model to parse all remaining in-domain data excluding those sentences for which we already have a human-annotated gold tree, and then marked the top tree for each sentence as pseudo-gold in the same way as above. We only show results here for appending the gold data to the self-trained data; without doing this, performance was slightly worse.

The results we see have much in common with those in Table 4.11. The self-training reliably gives a significant improvement in F-score of 1–1.5% against the variant using DUPLIC without self-training. Compared to the CONCAT baseline, the overall effect of the combination of DUPLIC and self-training is even more dramatic, with 1.6–1.8% improvements, corresponding to reductions in error rate of between 10 and 15%. Again, the results for exact match are mixed. We never get significant improvements over using DUPLIC alone; at best, there is a weakly significant improvement compared to CONCAT. The impact on Acc_{10} is always negative compared to DUPLIC, and in most cases is lower than the CONCAT baseline as well.

Compared to having no out-of-domain data, we might expect that the self-training when we have some in-domain data would be more effective, as we can parse in-domain data using a better parse selection model which is partially domain-tuned. On the other hand, this better initial model gives us a higher baseline and we already have more domain-tuned data available, so we might not be able to achieve as much using a noisy technique like self-training. These results provide some weak evidence to suggest that the latter effect is more important here, since the improvements against the variants without self-training data are more modest than when we are using entirely out-of-domain training data, although we would need a more comprehensive investigation to evaluate this thoroughly.

Both sets of results indicate that self-training produces parse selection models which have slightly different qualities to those produced using only hand-annotated data. The EDM scores show reliable improvements, suggesting that the models are better at getting the highest-ranked parse tree mostly correct. However the exact match scores show very little change, so it is of little assistance in getting the top-ranked tree exactly correct. Additionally, the self-trained models are noticeably worse at getting the correct tree into the top-10, and, presumably the top-500. So, when our aim is to get an exact-match parse tree somewhere in the top N, rather than to get the semantics as correct as possible, using self-training is unlikely to help and could in fact be counterproductive. This is particularly interesting given that the parse selection models are trained using full syntactic derivation trees (as described in Section 3.3.5), which only indirectly related to the semantic representation on which the EDM scores are based. If we are building a model for treebanking, it is probably not helpful to use self-training in this particular form, but if we simply want to use the parser outputs in downstream applications, self-training is a sound strategy if we have little or no in-domain data, since it only take a small amount of CPU time.

There are a number of ways the self-training procedure could be optimised. A new grid search could be useful, in case different maximum entropy parameters would be optimal. It is also possible that different volumes of training data (e.g. parsing more sentences to get pseudo-gold trees) could improve performance, and there could also be benefits from using different methods to combine the sources of training data. Another variant might be to vary the positive and negative synthesised annotations. For example, it may be productive to exclude some of the higher-ranked (but not top-ranked) trees entirely, so they do not contribute to the negative instances when there is a good chance that they correspond to well-formed parses. Nonetheless, as a first pass it shows the technique is useful, at least when we wish to use the parser output in external applications.

4.5 Discussion

4.5.1 Domain Adaptation Strategies

One of the important aspects of this chapter to other users of precision grammars is to suggest a strategy for achieving optimal performance over data in some new domain, and make a decision about how much effort to expend treebanking, and how best to make use of treebanked data. On the basis of this work, we would make the following recommendations:

- Unsurprisingly, out-of-domain data is much better than no data. If there are insufficient resources to treebank *any* in-domain data, one can expect tolerable parsing accuracy from using only out-of-domain data – the domain adaptation

performance penalty does not make the outputs from the ERG unusable (and we might expect this to hold for similarly constructed grammars).

- We can obtain modest improvements, at least in dependency F-score, by parsing a small corpus of in-domain data and using the top-ranked tree as pseudo-gold in a self-training strategy, although this is unlikely to give us improvements in exact-match accuracy.

- Further to this, the effort required to treebank around 11,000 tokens (750–850 sentences for the corpora here) gives substantial gains in accuracy compared to the benchmark of using only out-of-domain data – these 750 sentences are extremely valuable, and provide improvements in both exact match and dependency F-score. The time requirements for this are modest: using Redwoods machinery, Zhang and Kordoni (2010) found that it was possible to treebank a curated WSJ subset at 60 sentences per hour, while Tanaka *et al.* (2005) found that Japanese speakers could treebank 50 sentences of Japanese dictionary definitions per hour. So even with a conservative figure of 40 sentences per hour, 750 sentences would be under 20 hours of annotation time.

- Even simply concatenating 11,000 tokens of in-domain data to existing training data gives a good performance boost, but by applying the optimisation strategy using cross-validation we have discussed for DUPLIC, it is possible in some corpora to obtain accuracies close to those you would expect if you had 11,000 more training tokens (This echoes the result of McClosky *et al.* (2010) that careful selection of weights for combining could improve the accuracy of a treebank parser, although the optimisation method is cruder). Without performing the optimisation step, upweighting the in-domain corpus by a factor of 5-10 provides near-optimal performance across the corpora we examined. Augmenting this domain-tuned model with self-training data also provides a small additional boost in dependency F-score on top of this.

- If you have resources to treebank 23,000 tokens (roughly 1600 sentences) in total, you can achieve additional boosts in performance, although the value is considerably reduced.

- Beyond 23,000 tokens of training data for some domain, the gains in accuracy per treebanked sentence are more modest, so the effort would only be justified for more difficult domains or if maximising accuracy is of utmost concern.

4.6 Summary

In this chapter, we examined the impact of domain on parse selection accuracy in the context of precision parsing, evaluated across exact match and dependency-

based metrics. Our expectations from related work were that parse selection accuracy would be significantly improved by in-domain training data, and our findings confirm this. The domain match is particularly important if we are interested in returning a completely correct parse, although this is largely an effect of the greater fluctuations in the exact match metric. Also unsurprisingly, we found that in-domain training data is considerably more valuable in terms of accuracy obtained from a given number of training sentences, and we quantified the size of this effect.

We also explored ways to avoid the cross-domain performance penalty in parse selection. We found that the construction of even small-scale in-domain treebanks, which is fairly tractable, can considerably improve parse selection accuracy, through combining the in-domain with out-of-domain data, and we compared various strategies for doing so. We showed that linear combination of models from different domains can provide slightly improved performance compared to training from a monolithic concatenated corpus, although without careful selection of weights, it can also decrease. A better strategy for tuning a model to a domain with a small training corpus was to duplicate this small corpus some integral number of times. A multiplier of 5–10 often produces good results for the data we have shown here, but we have also shown that the optimal value for this parameter can be estimated on a case-by-case basis by using cross-validation over the training corpus, as the values are highly correlated.

Finally, we conducted a preliminary investigation into self-training for the ERG as another strategy to avoid cross-domain performance penalties, but requiring no labour at all from human annotators. This involved parsing a sample of in-domain data with an out-of-domain model, then treating the top-ranked parse as gold. We found that this was a somewhat viable strategy for domain adaptation, with significant improvements in dependency scores in the corresponding models compared with the best non-self-trained configurations using the same training data. However, the improvements in exact match were very modest, while in top-10 match we often saw a performance drop.

The findings of this chapter are significant for grammar consumers, who wish to have some idea of the performance penalty they can expect for parsing out-of-domain data, and more importantly strategies that can be applied to avoid this.

In the following chapter we move on to address a similar problem of wishing to create a parse selection model for a new domain, but consider the situation where there is some external treebank resource which is superficially incompatible. We evaluate whether it is possible to mine this resource for information to assist with domain adaptation for parse selection.

Chapter 5

External Resources for Parse Selection and Treebanking

5.1 Introduction

In the introduction to Part II we discussed the dimensions in which treebanks can be suboptimal for some particular target situation. Chapter 4 focussed on quantifying and overcoming the effects of a domain mismatch. In this chapter we explore ways to overcome another possible kind of incompatibility of a source treebank — it may not match the syntactic formalism of the grammar we wish to use for the target domain. That is, the assumptions underlying the syntactic trees (or dependencies) which make up the treebank may differ from those in the grammar or parser we are using. It might superficially appear in this situation that the incompatible treebank is of no use. However, as we noted in Section 2.7, linguistic information from superficially incompatible formalisms can still provide useful information in various ways. In particular, if a corpus is attractive for some other reason, such as being matched to the domain, but there is a formalism mismatch, we may be able to extract considerable useful information. This can give us an extra way to take advantage of annotation which has already occurred to maximise performance for some task while minimising the requirement for additional human labour, since some of the annotation work has already been done, even if it cannot be used in its original form.

We assume here that there are already syntactic annotations available from an external phrase structure treebank, and we wish to make the maximal use of this information on a precision HPSG-based grammar. We also assume that there is another treebank available which is compatible with the target grammar, but which does not match the domain in which we are interested. As such, this chapter explores the intersection between domain adaptation and formalism adaptation.

We investigate whether we can use these pre-existing partially incompatible annotations to our advantage when we already have a grammar. We may be able to

use this linguistic information to constrain the parse forest, which can be useful for two reasons. Firstly, this partially-constrained parse forest could be used directly to create an improved parse selection model for the domain of the external treebank, since a constrained parse forest is exactly what we use to train parse selection models, as discussed in Section 3.3.5.

Secondly, a partially-constrained parse forest could also reduce the requirements for expensive treebanking labour. If we wish to manually create an HPSG-based treebank to mirror the external phrase structure treebank, there is another closely related way we can take advantage of this information. In Redwoods-style treebanking, as discussed in Section 3.3.4, human treebankers manually apply constraints to reduce the size of the parse forest. If we can use the external treebank to apply at least some of these constraints automatically, we may be able to substantially reduce the cost of the treebanking process.

In this chapter, we examine both of these techniques. They are applicable in principle to any grammar compatible with the [incr tsdb()] treebanking machinery, using any external phrase structure treebank, although the approach could be more or less successful depending on the assumptions underlying the grammar and treebank and a range of other factors.

Following the theme of this thesis, the HPSG-based grammar we use for these experiments is the ERG.[1] The external phrase-structure treebank is the GENIA treebank (GTB) discussed in Section 3.4.2, so the domain we are targeting is that of biomedical abstracts. The methods are mostly not domain-specific so could be applied to any treebank compatible with the Penn Treebank, (PTB, which we first described in Section 2.4.1), including the PTB itself. However, we had an interest in parsing the domain of biomedical abstracts for external reasons — in particular for using in the downstream system described in Chapter 6. This meant that the GTB was an obvious choice as a case study, although other biomedical treebanks would probably have been similarly useful.

5.2 Treeblazing

In this chapter, we are investigating the process of constraining an HPSG parse-forest using an external treebank. Tanaka *et al.* (2005), as described in Section 2.7, applied a similar procedure to the Japanese grammar JaCy using gold-standard POS-tags, and denoted this process *blazing*, from a term in forestry for marking trees for removal. Our work is an approximate syntactic analog of this procedure, but since our gold-standard external annotations are parse trees rather than POS-tags, we denote this process *treeblazing* (or simply use the shorthand term "blazing").

[1] All work in this chapter uses Subversion revision 8990 from `http://svn.emmtee.net/erg/trunk`. This has minor lexicon changes compared with the '1010' tagged version used in Chapter 4.

The constraints are applied by disallowing certain candidate HPSG trees from a parse forest on the basis of information derived from the external treebank. The goal is not to apply all constraints from the source treebank to the parse trees from the target grammar; rather, we want to apply the *minimal* amount of constraints possible, while still sufficiently restricting the parse forest for our target application.

To run an iteration of treeblazing, we take the raw text of each sentence from the external treebank and parse it with the target grammar, to create a parse forest populated with the best trees according to some pre-existing parse selection model. The size of this parse forest[2] is capped at some threshold, which is conventionally 500 trees. The [incr tsdb()] treebanking machinery computes the set of discriminants from the target grammar (as discussed in Section 3.3.4) corresponding to a meaningful difference between the candidate derivation trees. This occurs in exactly the same way as if they were to be supplied to a human treebanker. Each of these discriminants is supplied to the treeblazing module.

The treeblazing module has access to the phrase structure tree from the source treebank which it compares with these discriminants. Each discriminant includes the name of the rule or lexical entry, as well as the corresponding character span in the source tree. The module then compares the source tree and the discriminants, and decides what the status should be for each individual discriminant. In principle, any discriminant could be ruled out, ignored or asserted to be true, in the same way as if a human were manually treebanking the sentence. However, the treeblazing module only ever rules out or ignores discriminants, since it is relatively easy to determine whether a discriminant is incompatible with the GTB analysis (by looking for conflicts), and harder to know for certain on the basis of the GTB analysis that a particular discriminant should be asserted to be true. [incr tsdb()] performs inference to rule out several trees on the basis of these decisions in combination, so providing it with several ruled-out discriminants enables it to decide on a smaller subset of non-conflicting trees in any case, and means that the blazing module is not required to make the comparatively more difficult decision to assert that a discriminant is true.

Primarily, the blazing module evaluates whether a discriminant is a *crossing-bracket discriminant*, i.e. corresponds to phrase structures in the ERG derivation trees which would have crossing brackets with any overlapping constituents. If this is the case, the discriminant conflicts with the GTB analysis, and it is ruled out. It is also possible to determine conflicts based on the name of the lexical type, as we discuss in Section 5.2.2. In cases where no conflicts are detected, the discriminant is simply ignored.

The decisions on these discriminants are returned to [incr tsdb()] which infers which corresponding analyses should be ruled out on the basis of this information.

[2]Throughout this chapter we use the term *parse forest* as an informal shorthand to refer to a group of parse trees for a particular sentence, rather than using it to imply that the trees are stored in some packed representation, which is how the term is used in other work such as Zhang *et al.* (2007)

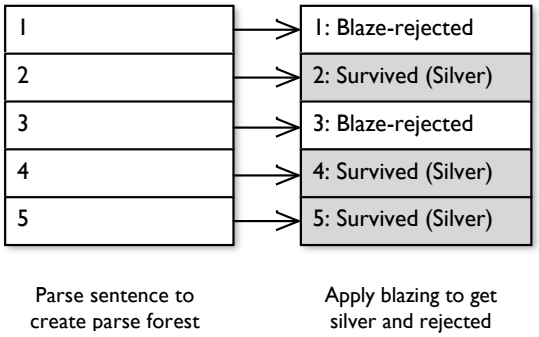


Figure 5.1: An example of usably blazing a sentence, rejecting some trees while some silver trees remain

This process happens with all discriminants for a sentence simultaneously, so it is possible to rule out all parse trees. This may indicate that none of the candidate parses are desirable, or that the imperfect blazing process is not completely successful. We call the set of trees remaining after blazing *silver* trees, to represent the fact that they are not gold standard, but are generally of better quality than the rejected analyses.

If the blazing process successfully rejects some trees from the forest to create a subset of silver trees, we refer to the sentence as *usably blazed* since the blazing process has added information, and this is the ideal situation, which we show diagrammatically for a parse forest of only five trees in Figure 5.1. There are two reasons that a sentence could fail to be usably blazed. One possibility is that the blazing module could be unable to reject any discriminants, which we call *unblazed*. This would result in no corresponding parse trees being rejected. Alternatively it could *overblaze*, or reject several discriminants simultaneously which are incompatible, so the intersection of the sets of corresponding trees is empty. Both of these failure possibilities are represented in Figure 5.2, and these cases are explored in more detail in Section 5.4.5.

5.2.1 Blazing ERG Trees using the GENIA Treebank

Here we are using the GENIA treebank and the ERG as exemplars of a suitable treebank and grammar pairing. We use the procedure as described above: for each GTB sentence, [incr tsdb()] computes discriminants for the top-500 parse forest from the ERG using some pre-existing parse selection model and supplies them to

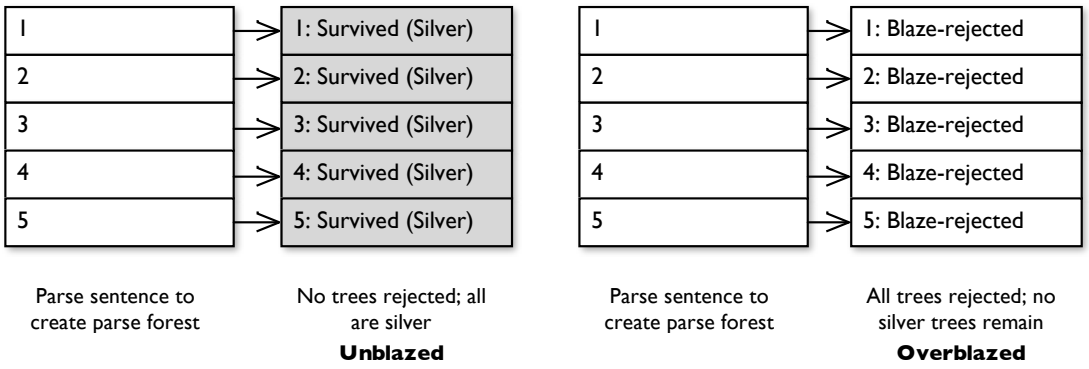


Figure 5.2: The two cases of blaze failure: *unblazed*, where no trees are rejected and *overblazed*, where all trees are rejected

the blazing module. This module has access to the GTB XML source of the corresponding sentence and compares this source with the discriminants to decide which discriminants to reject on the basis of those which would cause crossing brackets with particular constituents from the GENIA tree. The GTB constituents are matched with ERG discriminants on the basis of character spans, which are included in the discriminants and can be easily calculated for the GTB tree. The broad details are in principle applicable to a wide range of such pairings, but there are necessarily customisations which are required in each case, which we describe here for this particular case.

5.2.2 Blazing Configurations

Both the GTB and the ERG are created with some kind of basis in theoretical linguistics, so we would expect many of the assumptions underlying the phrase structures to hold true for both, particularly for phenomena such as PP-attachment and co-ordination. However there are disparities, even between the unlabelled bracketing of the GTB and ERG trees.

These can occur for several reasons. Firstly, the ERG is a precision grammar which produces deeply-nested analyses, while the GTB is comparatively shallow. This is not such a problem here since we are purely using the GTB to constrain the ERG analyses, not vice versa, but it does mean that if we want to greatly reduce the set of possible trees, we may need to artificially add extra structure to the GTB. Additionally, there are systematic differences in the handling of certain very common

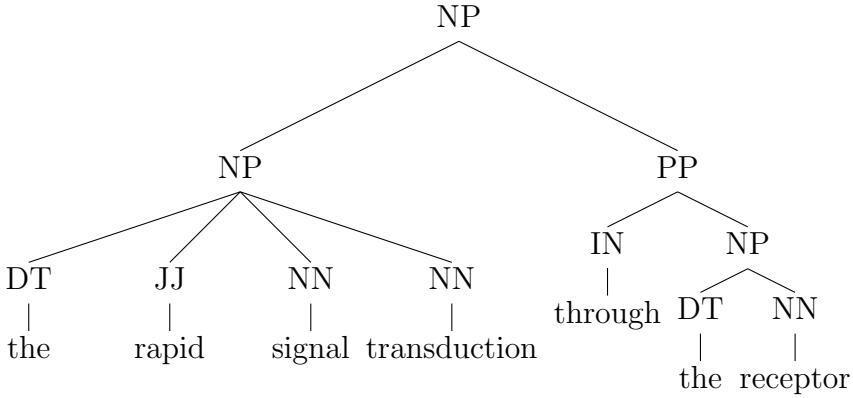
phenomena – for example, the ERG has binary branching (at most two children per node) instead of the multiple branching in the GTB. These can be handled by either avoiding the corresponding constituents or attempting to remap to a more acceptable structure. The variation in the means of handling these account for much of the variation between the blazing configurations described below.

Another very important pervasive and systematic difference is the attachment of specifiers and pre- and post-modifiers to noun phrases. The GTB attaches pre-modifiers and specifiers as low as possible, before attaching post-modifying PPs at a higher level, while the ERG makes the opposite decision and disallows this order of attachment. We show a sample noun-phrase exemplifying the differences in modifier attachment and branching in Figure 5.3

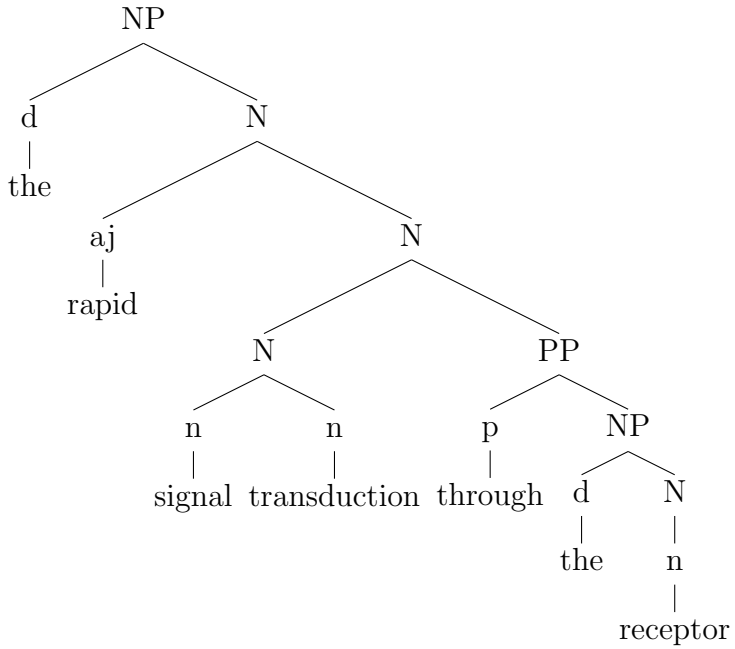
One strategy for handling this is to make as few assumptions as possible while avoiding spurious conflicts. Denoted **IgnoreEqPar** or **IEP** for *ignore equal parent*, it involves ignoring GTB nodes with the same label as the parent when looking for constituents with crossing brackets. From the GTB tree in Figure 5.3(a), the blazing module would ignore the boundaries of the lower-level NP (*the rapid signal transduction*) when looking for crossing-bracket discriminants. This ensures that we never rule out the corresponding good subtree shown in the figure in favour of some invalid bracketing from the ERG that by chance happens to have no conflicts with the GTB tree; meanwhile the PP in the example would still be considered. Note that in the case of a flat NP with no post-modifiers, no changes are necessary, since the external boundaries still correspond with the edges of the top-level NP in the ERG, and the extra internal boundaries within the ERG will have no effect since they cannot cause any crossing brackets.

An alternative strategy is **RaisePremods** (or **RP**). In this strategy, to avoid discarding possibly valid syntactic information, we attempt to account for the systematic differences by mapping the GTB as closely as possible to the matching structures we would expect in the ERG before looking for crossing-bracket discriminants.

As noted in Section 3.1.1, the rules in the ERG are unary-branching or binary-branching, so each node in the derivation trees it produces has at most two children. Because of this, the necessary first step for **RaisePremods** is binarisation. This is performed in a similar way to much previous work (Xia 1999; Miyao *et al.* 2004; Hockenmaier and Steedman 2002) which was explained in Section 2.4.2. This binarisation applies to all phrasal categories, not just NPs. We heuristically determine the head child of each phrase using a head percolation table very similar to the one described in Section 2.4.2. More specifically, the head of an NP must have a POS tag starting with 'NN' and head of a VP must have a POS tag starting with 'VB', along with some less important corresponding mappings for adjectives and adverbs. Additionally, each phrasal category itself is also considered as a head of a parent with the same phrasal category. We examine each candidate phrase labelled 'NP', 'VP', 'ADJP' or 'ADVP', and if the node has more than two children, we attempt to determine the node containing the phrasal head by using the above mapping. If the

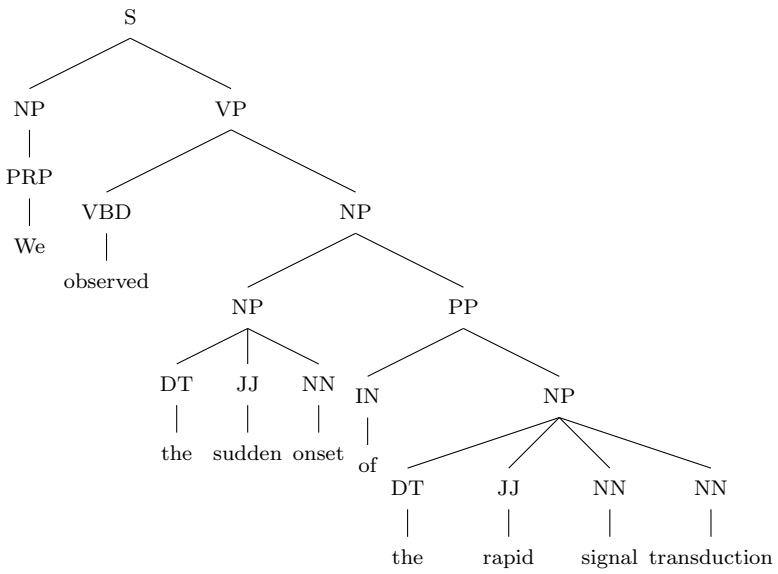


(a) A sample native GTB subtree



(b) A simplified ERG subtree

Figure 5.3: Sample trees



(a) Original tree

Figure 5.4: A (fictitious) sample GTB tree, continued in Figure 5.5

head node is at the right extreme (which is generally the case for English), we binarise by creating a new node which is the parent of the rightmost and second-rightmost node, and repeat this until the parent has only two immediate children. The new nodes copy their label from the parent node, except if the parent is 'NP', when the child is labelled 'Nbar', to indicate a phrase-internal nominal element.[3] The process is deliberately conservative; if we cannot determine a single canonical head node (which would occur for noun compounds, where there are multiple members labelled 'NN'), no changes are made to the phrase. Pseudocode for **RaisePremods** is shown in Appendix B.

We now have ERG-style binarisation, but the determiners and modifiers are still arranged differently to the ERG. To normalise these to the ERG style, we systematically alter the attachment positions of determiners and pre-nominal modifiers, forcing them to attach as high as possible, while preserving the binary branching. As a lightweight but imperfect attempt to avoid creating invalid structures for NPs involving apposition and conjunctions, this NP rearrangement is abandoned if any tokens at a given level are parentheses, commas or conjunctions.

In Figure 5.5, we show an example of this as applied to the tree in Figure 5.4 in two stages — firstly binarisation (Figure 5.5(a)) and then raising premodifiers (Figure 5.5(b)). Note how we are not able to canonically determine the head of the

phrase containing the noun compound *signal transduction* (since there are multiple occurrences of 'NN'), so no changes are made to that part of the tree.

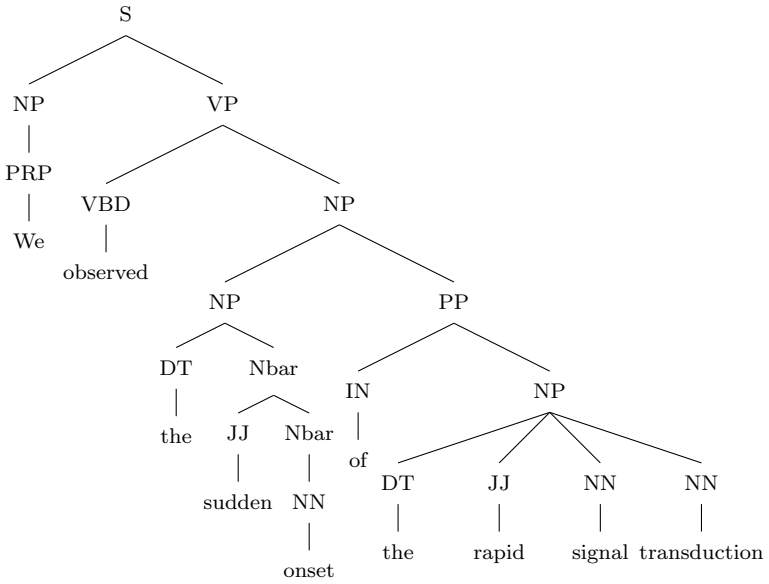
Another rule concerns the internals of noun compounds, which are flat in the GTB; we may wish to add some structure to them. As discussed in Section 5.3.1, biomedical noun compounds are predominantly left-bracketed, and as we note later in Section 5.3.1, left-bracketing is also a tie-breaking policy for annotating the test set. In the **BracketNCs** strategy (*bracket noun compounds*), we added bracketing to noun compounds to have noun sequences maximally left bracketed, and adjectives attaching as high as possible. This is of course making assumptions which are not explicitly licensed by the data (as well as arguably overfitting to our data set and its annotation guidelines), so this transformation is something we evaluate only when we are unable to get sufficient constraints from the less restrictive approaches (either on a global or per-sentence level), as we describe in Sections 5.4.5 and 5.5.1.

We also use a mapping strategy which does not make changes to the tree structure but which uses the POS labels to rule out trees, denoted **MapPOS**. It uses the prefixes of the lexical types — e.g. a simple transitive verb would have the lexical type V_NP_LE, where the prefix 'v' indicates 'verb'. We used a mapping constructed by manual inspection of a correspondence matrix between the POS tags produced by TnT (Brants 2000) and the lexical type prefixes from the gold-standard ERG parse of the same sentences over a WeScience subset. This gave us the matching ERG type prefixes for 20 PTB/GTB POS tags, which are mostly what we would expect for the open classes – eg *VB** verb tags map to the 'v' prefix. The full set of POS mappings is shown in Appendix B.

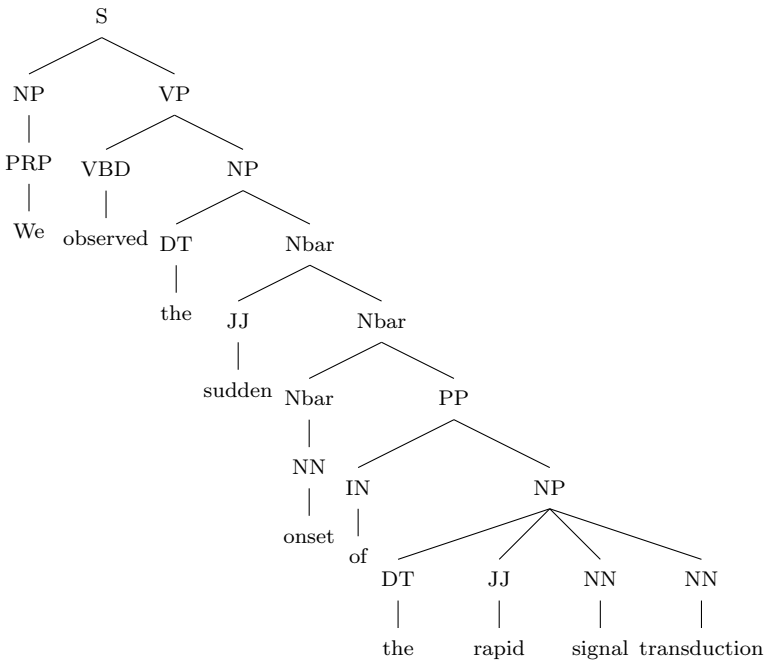
During mapping, given a pairing of a GENIA tree and a set of ERG discriminants, for each POS tag or inner node in the GENIA tree, we find all lexical discriminants with the same character span. If there are multiple discriminants with different matching labels, and there is at least one allowed and one disallowed by the mapping, then we reject all disallowed discriminants.

This is less sophisticated than the mapping technique of Tanaka *et al.* (2005). Firstly the construction of the mapping between the lexical types and the GTB POSs is somewhat more ad hoc. Additionally, this method does not have as sophisticated a method for handling POS-conversion in some morphosyntactic phenomena. For example a verb with an '-ing' suffix can be used syntactically as an noun, as in *the* ***finding*** *of the tomb*. Productive phenomena such as these are handled in the ERG by lexical rules applying to the original lexical item – in this case FIND_V1 – which allow the verb to fill the slot of an adjective phrase higher in the derivation tree. But the corresponding discriminant would refer to the original lexical entry, so it would appear to the blazing module that the discriminant was referring to a verb. However this token would be tagged as an noun according to the PTB/GTB guidelines, so

[3]In fact, these inner node labels are purely for human readability, as they are not used in the blazing procedure.



(a) Step 1: Binarisation applied



(b) Step 2: Premodifiers Raised

Figure 5.5: The first and second steps of applying the **RaisePremods** transformations to the tree in Figure 5.4

IgnoreEqPar	**IEP**	*ignore equal parent*: ignore GTB nodes with the same label as the parent
RaisePremods	**RP**	*raise premods*: binarise, then attach premodifiers high in NPs
BracketNCs	**BNC**	*bracket noun compounds*: in noun compounds, have noun sequences left-bracketed, and attach adjectives high
MapPOS	**MP**	*map POS*: map GTB POS to ERG lexical types
x + **y**		Strategies **x** and **y** are applied simultaneously
x, **y**		The first listed strategy **x** is applied and then the second strategy **y** is applied if the first strategy fails to usably blaze the sentence.

Table 5.1: Summary of strategies used for various aspects of blazing and ways of combining them.

the blazing module would make an incorrect decision in this, as there is another competing analysis available which superficially matches the POS of the GTB tree but is nonetheless incorrect. In this case, there is an ERG lexical entry for *finding* as a noun, which is incorrect in this case.[4] This would mean that when the blazing module saw the discriminant corresponding to a verb analysis (the correct one from above) along with one for a noun analysis, it would erroneously reject the verb analysis, since a preferred analysis is available. The architecture of the original Tanaka *et al.* method was in contrast able to take such conversions into account, which is not possible with the setup we use here. We can estimate how much of a problem this is likely to be by evaluating the figures over our development corpus, described later in Section 5.3.1. In that corpus, lexical rules which result in POS-conversion, and thus have the potential to cause an incorrect rejection, apply to only 1.6% of tokens, at an average of 0.33 occurrences per sentence. For a potentially problematic case to cause an invalid rejection, we also require the very specific circumstances described above of a competing incorrect analysis, which could only occur for a small number of lexical entries. It therefore seems justified to use this approximation.

All of the strategies from this section and their combinations are summarised in Table 5.1.

[4]The noun version of *finding*, FINDING_N1 would be used for phrases such as the *the findings of the inquiry*, where *of* denotes a possessive rather than a complement of the underlying verb as in *the finding of the tomb*.

5.2.3 Example of Blazing

As an end-to-end example of the blazing process, we consider the following GTB sentence (from the abstract with PubMed ID 10082134):

(5.1) *The transendothelial migration of monocytes was inhibited by an antibody to PECAM-1*

Note that this sentence was selected so that a full example was manageable, which means it is uncharacteristically short for the GENIA treebank. The corresponding ERG parse forest is correspondingly small (in fact, it is atypically small even for such a short sentence), with only five candidate parse trees, with trees 1–5 shown in Figures 5.6, 5.7, 5.8, 5.9 and 5.10 respectively. Even for such a short sentence, however, the derivation trees are cumbersome to display in full, so the sections which are identical between the trees and therefore not relevant here have been simplified.

Since there are so few trees, it is possible to summarise in prose the key differences between the candidate analyses of Example 5.1. The differences all relate in some way to handling of prepositional phrases for this particular example. Primarily there is the question of where the prepositional phrase *to PECAM-1* attaches. Trees 1 and 2 (Figures 5.6 and 5.7) have it modifying the verb phrase *inhibited by an antibody*, tree 3 (Figure 5.8) has it modifying the prepositional phrase *by an antibody*, while in trees 4 and 5 (Figures 5.6 and 5.7) it modifies the noun *antibody*. The other differences are in the interpretation of the preposition *by* in relation to the passive verb *inhibited*: whether it is a marker of the agent in the passive construction, or a locative preposition. This is the distinction between sentences such as *She was seated by the usher* (in the interpretation where the *usher* is actively showing people to their seats) and *She was seated by the aisle*. Trees 1 and 4 have the passive agent interpretation (note that the *by*-PP is different in each case), while trees 2 and 5 have the locative reading.

These differences are reflected in the discriminants corresponding to this parse forest, shown in Table 5.2. These discriminants would be presented to a human treebanker or (more relevantly for this discussion) supplied to the blazing module. Each discriminant is associated with a 'key' corresponding to the name of the rule or the name of the lexical entry in the case of lexical ambiguity (which is not shown here). Most importantly, each discriminant also has a character range associated with it, indexing back to the source text. This is the primary piece of information used by the blazing module, although the key is also used for the **MapPOS** strategy described above since it includes the lexical type of the token.

The textual representation is for human-readability here, and would also be used by a human annotator in the treebanking GUI, but is not used for the blazing process. Similarly, the token indexes correspond to those in the GENIA tree of the sentence in Figure 5.11 and are shown only for expository purposes relating to this example.

The matching actually occurs on the basis of character spans, rather than these token indexes. The main reason for this is that the tokenisation is in general different between the GTB and the ERG. Firstly, there are systematic differences, such as the way the ERG treats punctuation as affixes rather than stand-alone tokens. Additionally, the preprocessing of named entities which we discuss in Section 5.3.2 means that the ERG tokenisation may differ from that found in the GTB in other ways — indeed, the tokenisation can be different between competing ERG analyses.

We have also manually annotated the table with the corresponding trees, but this information would not be available to the blazing module — the only view it has of the data is the supplied discriminants (comprising keys and character ranges), and the inference about which trees match is performed by [incr tsdb()] after the blazing module has rejected some subset of discriminants.

The blazing module does have access to the corresponding raw, XML-encoded GTB tree. We show a representation of the corresponding tree in Figure 5.11. The token indexes have been added for human readability, but as noted above, the matching would usually occur on the basis of character spans, which can be straightforwardly calculated for the GENIA tree. It is the comparison of these inferred character spans with those provided as part of the discriminant which informs most of the blazing process.

The blazing module applies any appropriate transformations (e.g. **RaisePremods**) or rules for ignoring nodes (e.g. **IgnoreEqPar**) to the appropriate GTB tree, and then compares each remaining GTB node to every partially overlapping discriminant. If the discriminant character boundaries completely surround or are completely surrounded by the corresponding GTB constituent, no action is taken. Otherwise, we have crossing brackets — the discriminant has one boundary inside the constituent, and one boundary before or after (as noted above, punctuation characters are ignored here). In that case, we reject the crossing-bracket discriminant.

For this particular example, if we were using the **IgnoreEqPar** strategy, we would check the constituents shown in the leftmost column of Table 5.3 against any partially overlapping discriminants (listed in the second column). Following the rules described, we would subsequently determine the crossing-bracket discriminants to be those shown in the third column. The blazing module would therefore inform [incr tsdb()] that the union of those crossing-bracket discriminants (i.e. discriminants 4, 5 and 7) should be rejected, and [incr tsdb()] would infer in the usual way which parse trees would remain. From the annotations in the 'Trees' column of Table 5.2, we can determine which trees would be rejected at this point — the union of all trees excluded due to any individual discriminant. Discriminant[5] $4_{7:10}$ rules out tree 1, discriminant $5_{7:10}$ rules out tree 2, and discriminant $7_{8:10}$ rules out trees 1, 2 and 3 (the redundancy is not a problem here). The union of these is $1, 2, 3$, so after blazing we would be left with trees 4 and 5 as our silver trees.

[5]Subscripts on discriminants denote token spans, which have been added for readability.

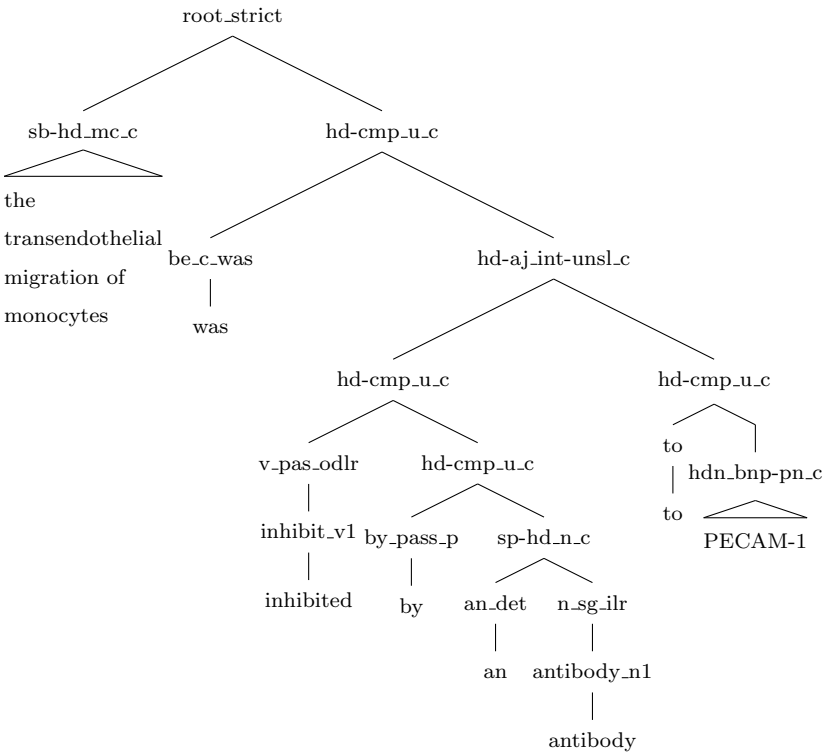


Figure 5.6: Derivation tree 1 for Example 5.1

Returning to our original description of the ambiguities in the tree, we have successfully determined that the *to*-PP attaches to the noun *antibody*, although we have not determined whether or not the *by*-PP denotes the agent of a passive verb. There is no guarantee that this process will leave the tree which a human annotator would select as one of the silver trees, although we would hope this would be the case most of the time. In this particular case, tree 4 is the tree that should be selected by a human treebanker, so the blazing process is at least partially successful. Ideally the only silver tree remaining would be the genuine gold tree, minimising the ambiguity, although this ideal is seldom achieved, as we discuss in Section 5.4.5.

5.2.4 Differences between Strategies

We might superficially expect the **IgnoreEqPar** and **RaisePremods** strategies to produce similar results, since they are both designed in different ways to handle the assumptions underlying the trees of the particular grammars, particularly with respect to noun phrases and to a lesser extent verb phrases. However, as we shall see later, both of them produce different results, with **RaisePremods** being substantially more restrictive in terms of producing a smaller set of silver trees from the same parse forest. Here we show an actual example of where this occurred. The sentence under

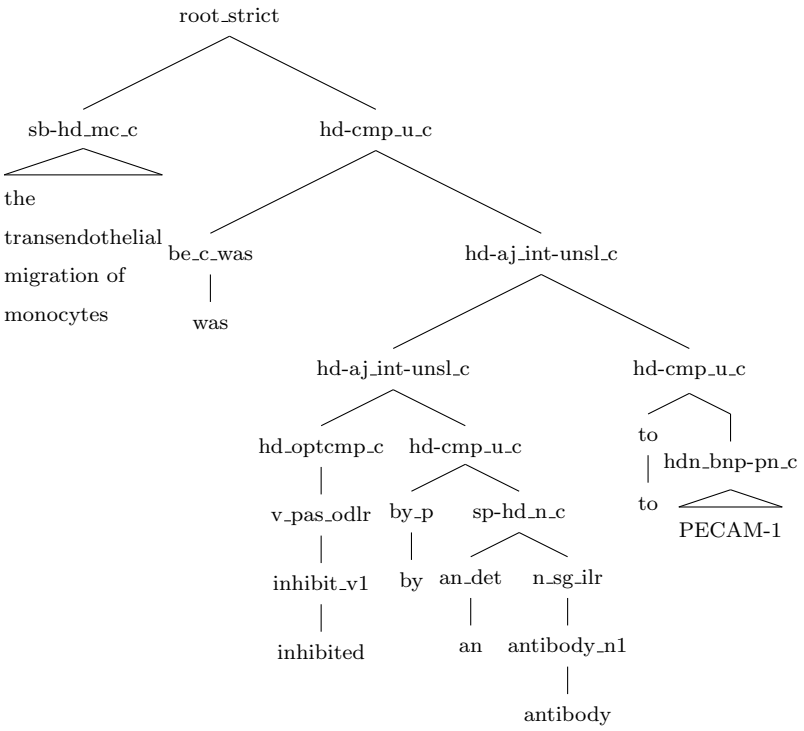


Figure 5.7: Derivation tree 2 for Example 5.1

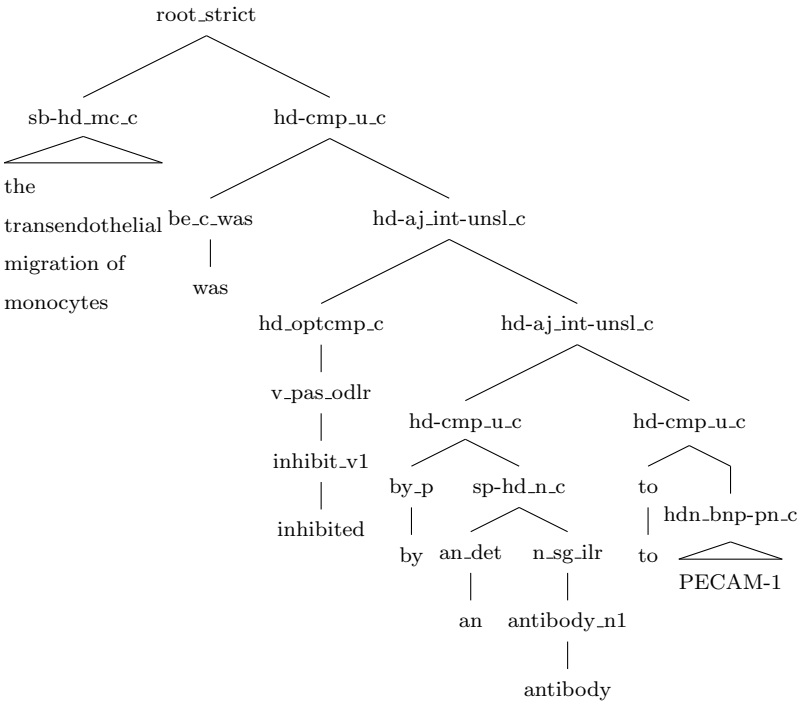


Figure 5.8: Derivation tree 3 for Example 5.1

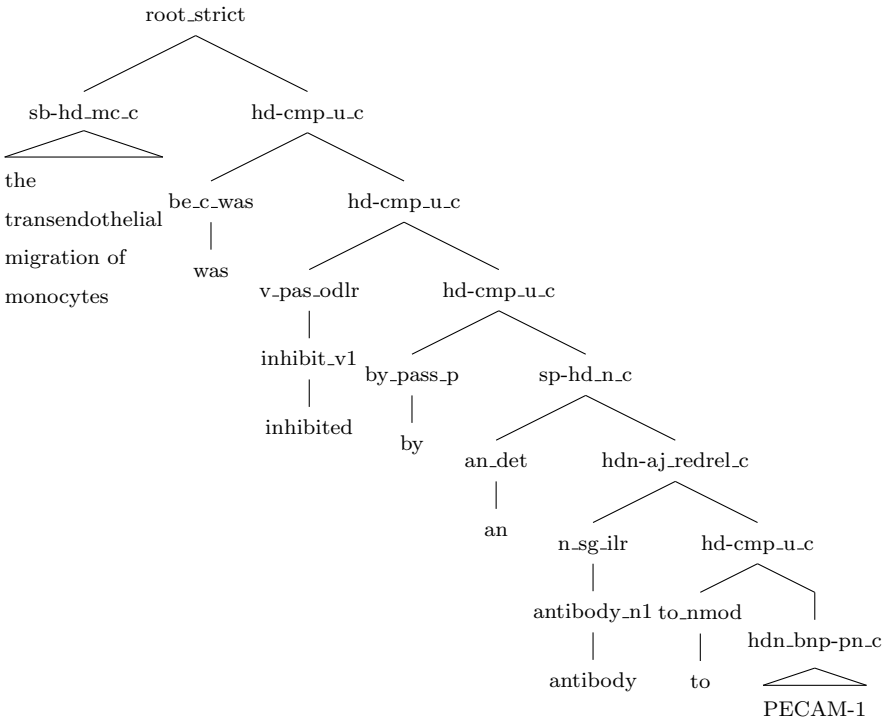


Figure 5.9: Derivation tree 4 (the correct tree) for Example 5.1

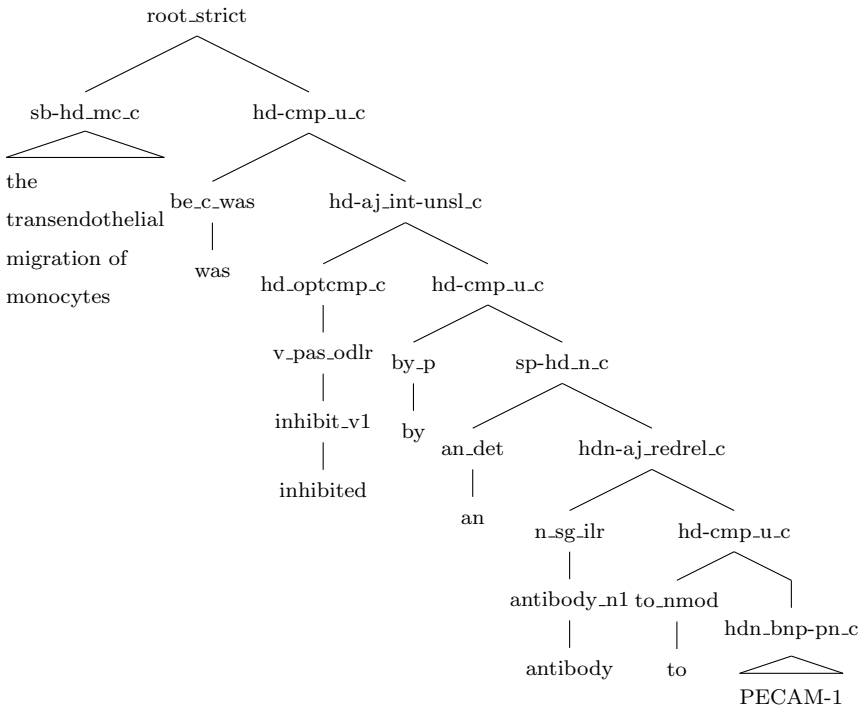


Figure 5.10: Derivation tree 5 for Example 5.1

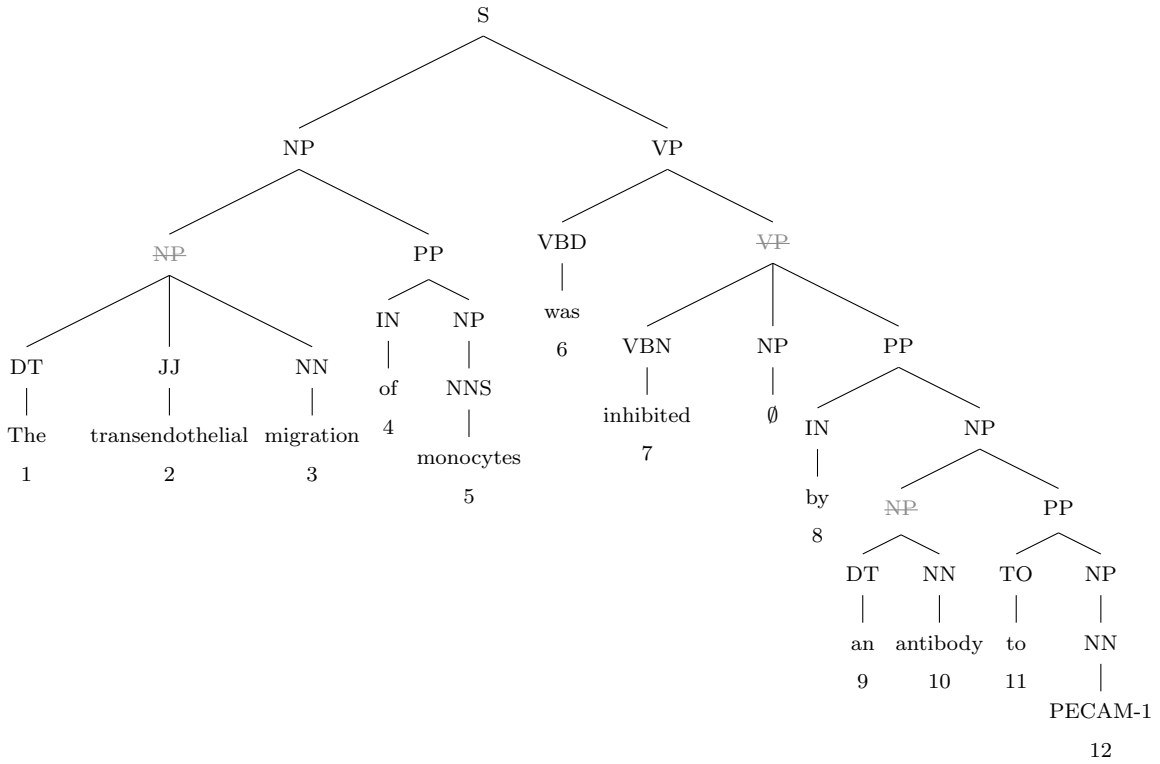


Figure 5.11: The corresponding GENIA tree for Example 5.1. For brevity the final full stop, a sister to the top level NP and VP, is omitted. Nodes that would be ignored by the **IgnoreEqPar** strategy are struck out and shown in grey.

ID	Chars	Tok.	Key	Representation	Trees
0	48:84	7:12	hd-aj_int-unsl_c	*inhibited by an antibody* || *to PECAM-1*	1, 2
1	48:84	7:12	hd-cmp_u_c	*inhibited* || *by an antibody to PECAM-1*	4
2	58:84	8:12	hd-aj_int-unsl_c	*by an antibody* || *to PECAM-1*	3
3	58:84	8:12	hd-cmp_u_c	*by* || *an antibody to PECAM-1*	4, 5
4	48:72	7:10	hd-aj_int-unsl_c	*inhibited* || *by an antibody*	2
5	48:72	7:10	hd-cmp_u_c	*inhibited* || *by an antibody*	1
6	61:84	9:12	sp-hd_n_c	*an* || *antibody to PECAM-1*	4, 5
7	58:72	8:10	hd-cmp_u_c	*by* || *an antibody*	1, 2, 3
8	64:84	10:12	hdn-aj_redrel_c	*antibody* || *to PECAM-1*	4, 5
9	61:72	9:10	sp-hd_n_c	*an* || *antibody*	1, 2, 3

Table 5.2: Discriminants offered by [incr tsdb()] for Example 5.1, to a human treebanker or the blazing module, showing the IDs, the half-open character spans (where the bottom index is included, and the top is excluded — these are what is used by the blazing module), the token index ranges (for human-readability only), the 'key' or rulename, a human-readable representation of the rule and the trees to which the positive discriminant corresponds (which the parse forest would be reduced to if the discriminant was positively selected by a human – this inference is performed by [incr tsdb()] but is not present in the discriminants). The blazing module uses only the range and, in configurations using **MapPOS**, the key.

Constituent	Overlapping Discriminants	Crossing-bracket Discriminants
$NP_{1:5}$	–	–
$PP_{4:5}$	–	–
$NP_{5:5}$	–	–
$VP_{6:12}$	$0_{7:12}$, $1_{7:12}$, $2_{8:12}$, $3_{8:12}$, $4_{7:10}$, $5_{7:10}$, $6_{9:12}$, $7_{8:10}$, $8_{10:12}$, $9_{9:10}$	–
$PP_{8:12}$	$0_{7:12}$, $1_{7:12}$, $2_{8:12}$, $3_{8:12}$, $4_{7:10}$, $5_{7:10}$, $6_{9:12}$, $7_{8:10}$, $8_{10:12}$, $9_{9:10}$	$4_{7:10}$, $5_{7:10}$
$NP_{9:12}$	$0_{7:12}$, $1_{7:12}$, $2_{8:12}$, $3_{8:12}$, $4_{7:10}$, $5_{7:10}$, $6_{9:12}$, $7_{8:10}$, $8_{10:12}$, $9_{9:10}$	$4_{7:10}$, $5_{7:10}$, $7_{8:10}$
$PP_{11:12}$	$0_{7:12}$, $1_{7:12}$, $2_{8:12}$, $3_{8:12}$, $6_{9:12}$, $8_{10:12}$	–

Table 5.3: Constituents extracted from the sample tree in Figure 5.11, evaluated against the discriminants from Table 5.2. The blazing strategy in use is **IgnoreEqPar**, which is why $NP_{1:3}$, $VP_{7:12}$ and $NP_{9:10}$ are ignored. For the **MapPOS** strategy, individual tokens would also be considered.

consideration is shown in Example 5.2. In this example, several irrelevant trees and discriminants have been omitted for clarity and brevity, so that we can instead focus on those which behave differently under each strategy. A subset of the parse forest is shown in Figures 5.12, 5.13, 5.14 (the tree a human should select as correct), 5.15 and 5.16.[6] A subset of the discriminants corresponding to the ERG trees is shown in Table 5.4, with, as before, the token spans from the GTB trees inserted for readability.

(5.2) *Human myeloid differentiation is accompanied by a decrease in cell proliferation*

We first examine which trees would have been accepted by the **IgnoreEqPar** strategy. In Figure 5.17(a) we show the corresponding GTB trees for the strategy, with ignored nodes struck out and greyed out. The constituents which would be evaluated by the blazing module are shown in the table in Figure 5.17(b), and from this we can see that **IgnoreEqPar** would reject only discriminant $6_{5:8}$. From Table 5.4, we can see that this means that tree 2 (Figure 5.13) is ruled out, leaving the set of silver trees as 1, 3, 4 and 5 when using the **IgnoreEqPar** strategy.

[6]This excludes 5 trees which would have been ruled out by both **IgnoreEqPar** and **RaisePremods**.

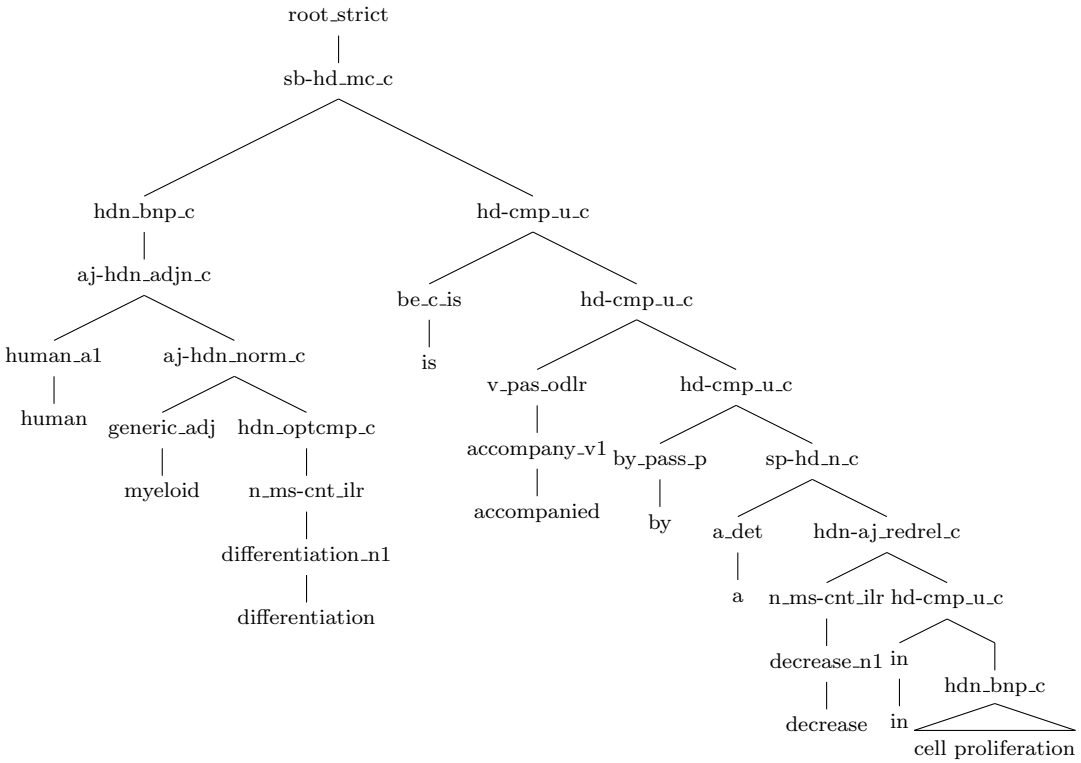


Figure 5.12: Derivation tree 1 for Example 5.2

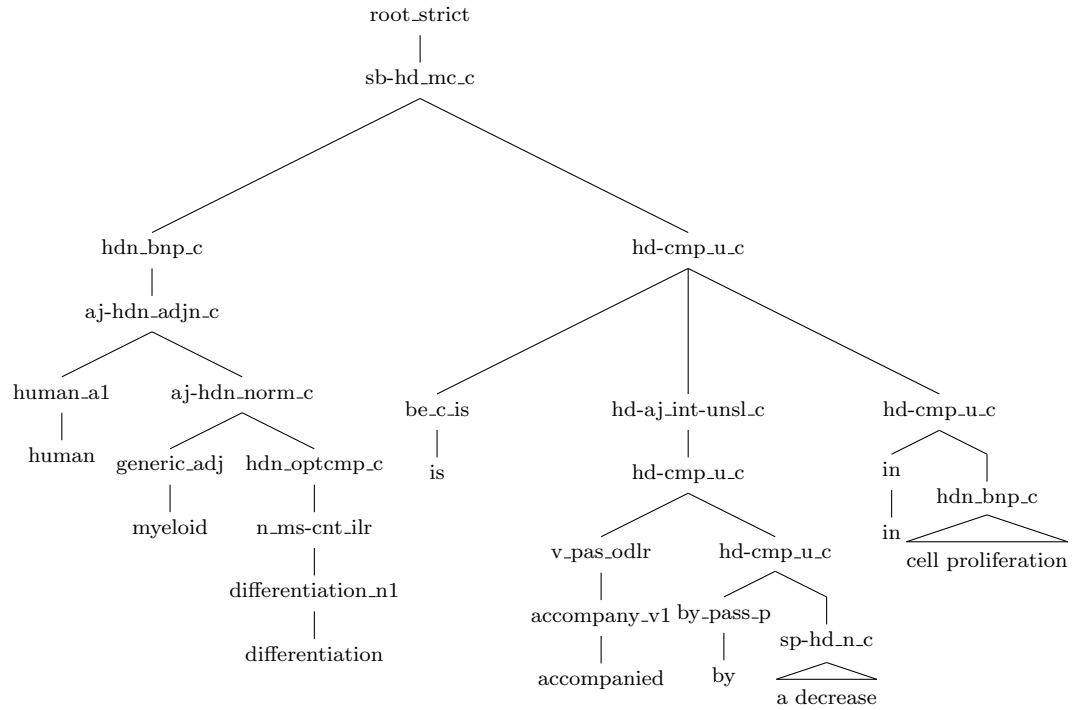


Figure 5.13: Derivation tree 2 for Example 5.2

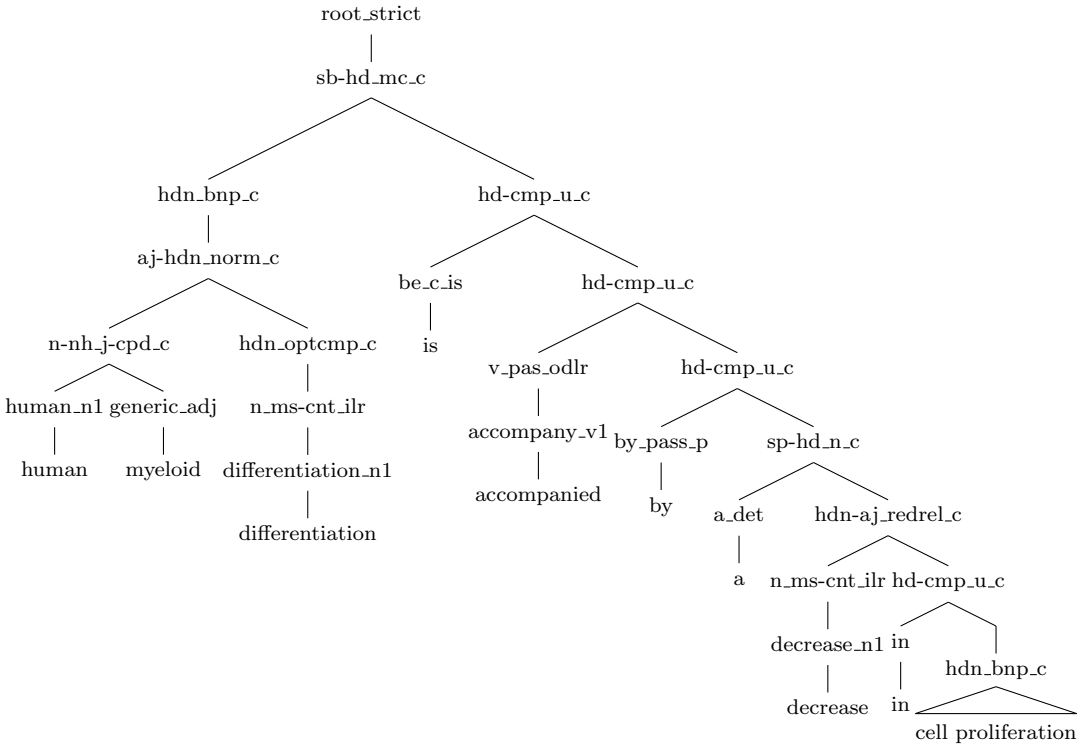


Figure 5.14: Derivation tree 3 (the correct tree) for Example 5.2

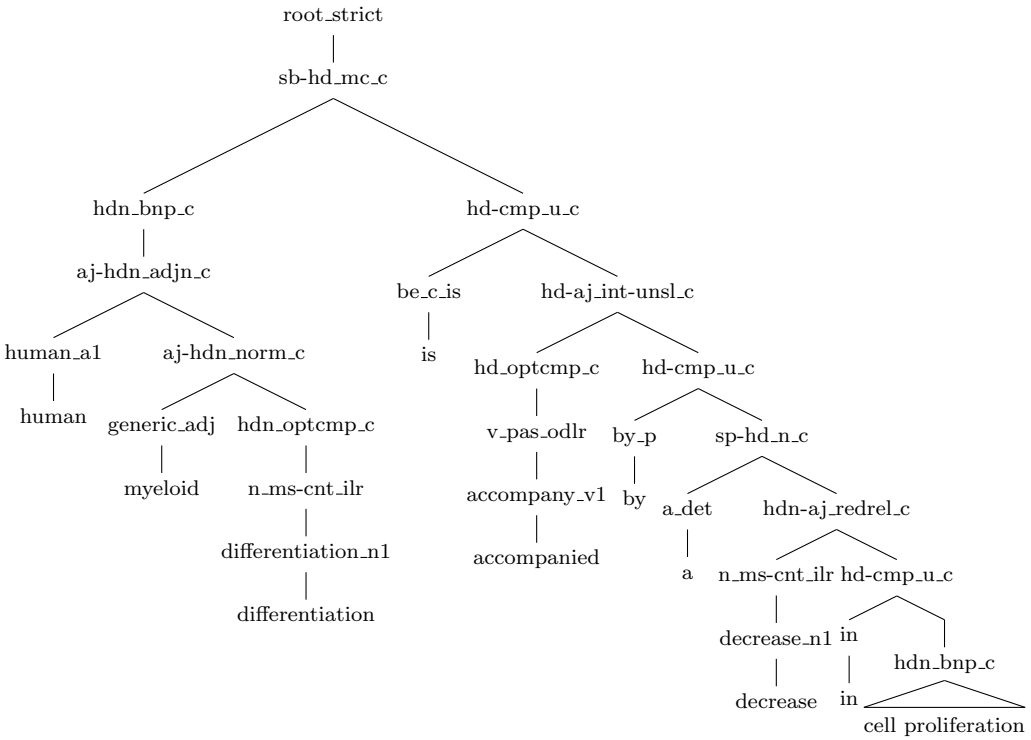


Figure 5.15: Derivation tree 4 for Example 5.2

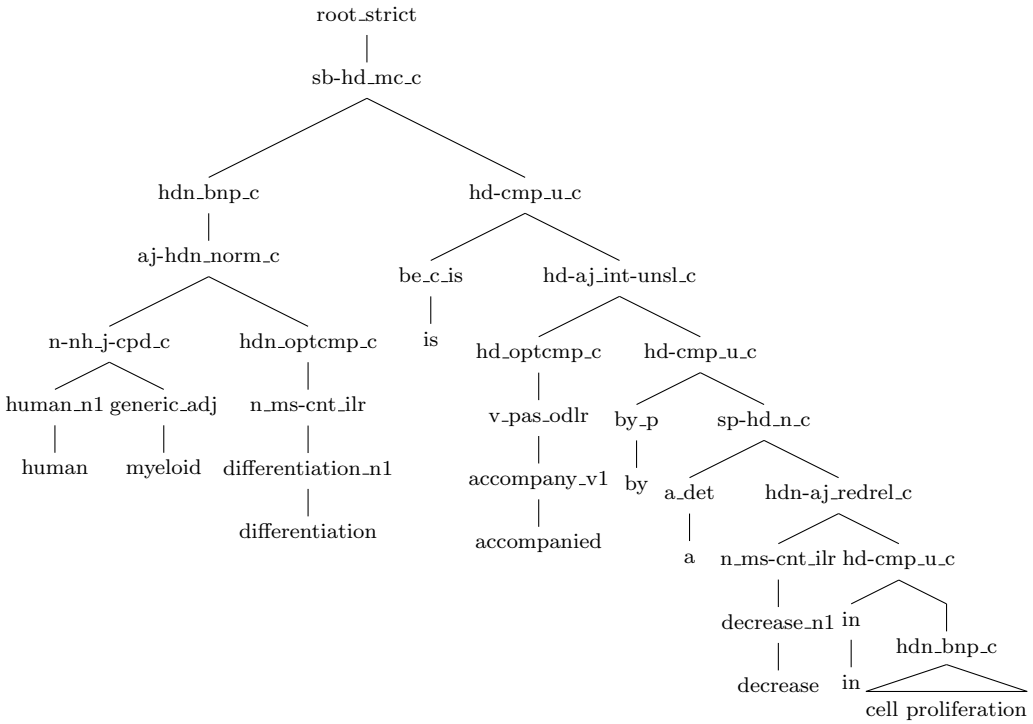
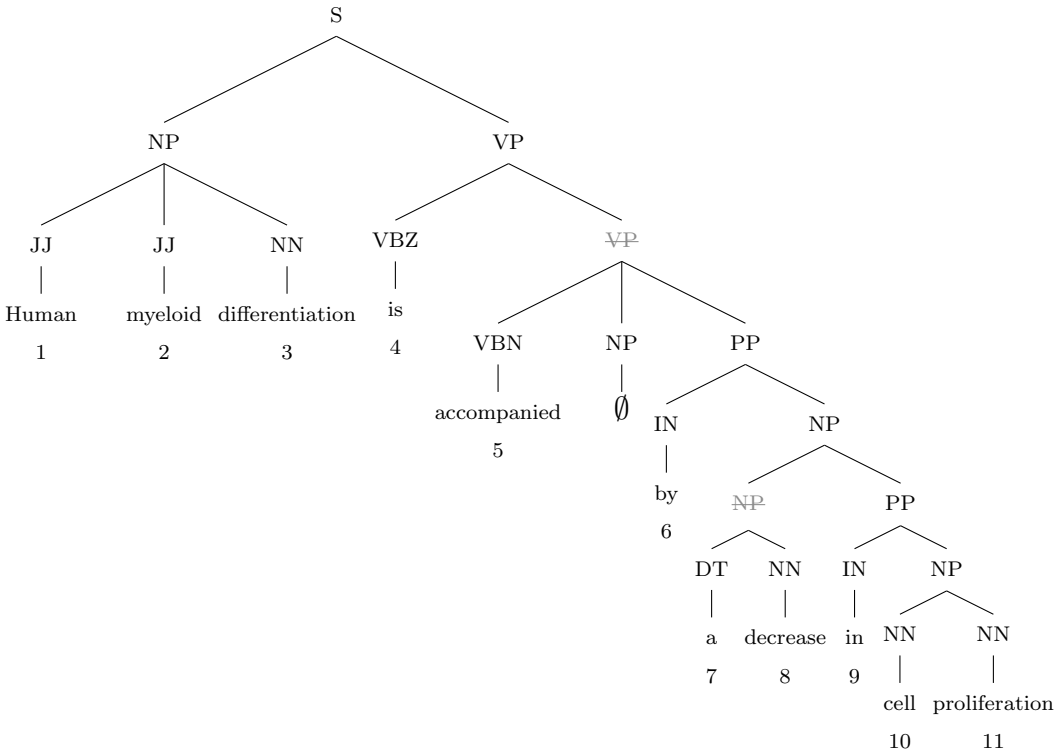


Figure 5.16: Derivation tree 5 for Example 5.2

ID	Chars	Tok.	Key	Representation	Trees
6	33:58	5:8	hd-cmp_c	*accompanied* || *by a decrease*	2
11	0:13	1:2	n-nh_j-cpd_c	*human* || *myeloid*	3, 5
13	48:58	7:8	sp-hd_n_c	*a* || *decrease*	2

Table 5.4: The relevant discriminants for our discussion corresponding to the trees in Figures 5.12 to 5.16 from Example 5.2, with token indexes added for human-readability.

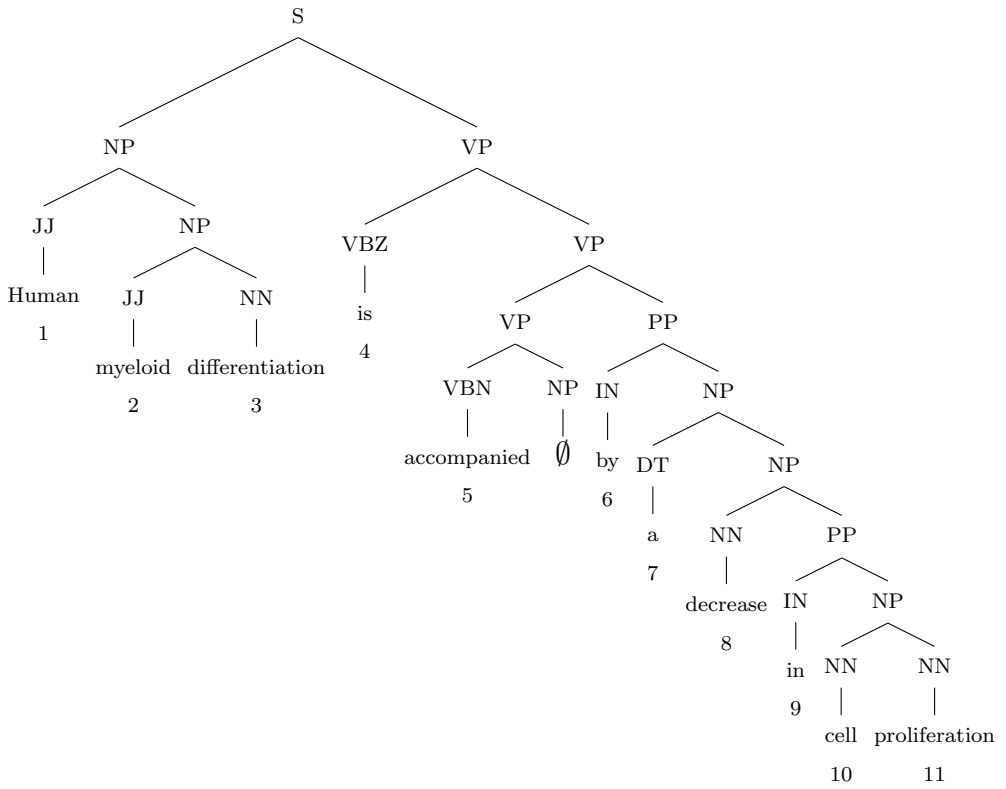


(a) GTB tree corresponding to Example 5.2 (no transformations, but struck-out, greyed nodes are ignored)

Constituent	Overlapping Discriminants	Crossing-bracket Discriminants
$NP_{1:3}$	$11_{1:2}$	–
$VP_{4:11}$	$6_{5:8}$, $13_{7:8}$	–
$PP_{6:11}$	$6_{5:8}$, $13_{7:8}$	$6_{5:8}$
$NP_{7:11}$	$6_{5:8}$, $13_{7:8}$	$6_{5:8}$
$PP_{9:11}$	–	–

(b) The constituents which would be extracted from the tree in Figure 5.17(a) and how they would evaluate against the discriminants in Table 5.4

Figure 5.17: Tree, applied constituents and corresponding discriminants for Example 5.2 using **IgnoreEqPar**



(a) GTB tree corresponding to Example 5.2 with **RaisePremods** transformations applied

Constituent	Overlapping Discriminants	Crossing-bracket Discriminants
$NP_{1:3}$	$11_{1:2}$	–
$NP_{2:3}$	$11_{1:2}$	$11_{1:2}$
$VP_{4:11}$	$6_{5:8}$, $13_{7:8}$	–
$VP_{5:11}$	$6_{5:8}$, $13_{7:8}$	–
$PP_{6:11}$	$6_{5:8}$, $13_{7:8}$	$6_{5:8}$
$NP_{7:11}$	$6_{5:8}$, $13_{7:8}$	$6_{5:8}$
$NP_{8:11}$	$6_{5:8}$, $13_{7:8}$	$6_{5:8}$, $13_{7:8}$
$PP_{9:11}$	–	–

(b) The constituents which would be extracted from the tree in Figure 5.18(a) and how they would evaluate against the discriminants in Table 5.4

Figure 5.18: Tree, applied constituents and corresponding discriminants for Example 5.2 using **RaisePremods**

Now we show how **RaisePremods** gives a different result for the same sentence. In Figure 5.18(a), we show the tree after it has been transformed according to the appropriate rules. The constituents and the corresponding cross-bracket discriminants are shown in the table in Figure 5.18(b). From here, we can see that discriminants $6_{5:8}$, $11_{1:2}$ and $13_{7:8}$ are all rejected in this case, which according to Table 5.4 means that the corresponding trees 2, 3 and 5 would all be ruled out, leaving only trees 1 and 4. Compared to **IgnoreEqPar**, **RaisePremods** is able to reject two extra discriminants. One of them, $13_{7:8}$, does not result in ruling out any additional trees, since it does not constrain the forest more that $6_{5:8}$. This is in fact what happens for a large number of extra discriminants rejected by **RaisePremods** – due to the redundancy in the discriminants, this extra information would also have been implicit in the discriminants rejected by the more conservative **IgnoreEqPar** strategy, even though there are fewer of these in general. However, the other discriminant $11_{1:2}$ rules out two additional trees, resulting in a more constrained parse forest. In this particular case, it happens to be that **RaisePremods** rules out tree 3, the tree which should be marked as a human annotator as correct, so in this case it is overzealous in the trees it rules out (however it differs from tree 1 only in noun compound bracketing, which is a notoriously difficult distinction to make in particular edge cases).

5.3 Working With Biomedical Data

We explore two branches of experimentation using a common core of tools, resources and methods. This section describes the test data we use, and some peculiarities of parsing biomedical data that affected our experiments.

5.3.1 Development Dataset

In order to evaluate the impact of the proposed method on parser accuracy over biomedical text, we require a gold-standard treebank in the target domain. We use a subset of the data used in the GTB, created by first removing those abstracts (approximately half) that overlap with the GENIA event corpus (GEC: Kim *et al.* (2008)), to hold out for future work. From this filtered set, our test corpus comes from the 993 sentences of the first 118 abstracts (PubMed IDs from 1279685 to 2077396). We refer to this treebank as 'ERGGTB'.

We treebanked this data using the Redwoods treebanking methodology (Oepen *et al.* 2004), where the treebank is constructed by selecting from a parse forest of candidate trees licensed by the grammar, as described in Section 3.3.4. The treebank annotators (the author of this thesis and Dan Flickinger, the chief architect of the ERG) both have detailed knowledge of the ERG, but no domain-specific biomedical expertise. As a proxy for this, we used the original syntactic annotations in the GENIA tree when a tie-breaker was needed for ambiguities such as PP-attachment

or co-ordination. We had access to the GENIA trees while treebanking, but deliberately only referred to them when the ambiguity was not resolvable on linguistic grounds. The first 200 sentences of the corpus were double-annotated in each round of treebanking (agreement figures for unseen data are shown in Section 5.5.2)

The first round of annotation of a subset of the corpus was to determine a suitable parser configuration and calibrate between annotators. Following this, we developed a set of annotation guidelines, which is shown in Appendix A. The primary domain-specific guideline relates to the treatment of noun compounds, which are never disambiguated in the flat GTB structure. In the biomedical domain, noun compounds are generally left-bracketed — 83% of the three-word compounds according to Nakov and Hearst (2005) — so we stipulated that noun compounds should be left-bracketed and adjectives attached high in cases of doubt, as a tie-breaking strategy.

We also used this first-iteration treebank to build a domain-tuned parse-selection model. We achieved this by duplicating the data 10 times and combining it with the WeScience corpus using the Duplic method described in Chapter 4, which was a robust performer for handling sparse in-domain data. We also made some changes to the parser's handling of named entities, as described in Section 5.3.2. We then reparsed the treebank with the new parsing configuration and parse selection model, giving 866 parseable sentences. After updating the treebank according to the new guidelines using this new parse forest, and checking inter-annotator agreement on the calibration set, we annotated the remaining sentences. All accuracy figures we report are over the data set of 669 trees that was complete at the time of experimentation.

5.3.2 Biomedical Parsing Setup

We parsed sentences using the ERG with pet, which uses POS tags to constrain unknown words as described in Section 3.3.6. Following Velldal *et al.* (2010), we primarily use the biomedical-trained GENIA tagger (Tsuruoka *et al.* 2005), but defer to TnT (Brants 2000) for tagging nominal elements, because it makes a useful distinction between common and proper nouns. The GENIA tagger tags these all as common nouns, which is largely due to a design decision in the tagset of the training corpus (Tateisi and Tsujii 2004), where it was noted that common and proper nouns were costly to distinguish, and unlikely to be useful for downstream applications in this domain.

Biomedical text poses a particular set of challenges compared to many other domains, mostly relating to named entities, such as proteins, DNA and cell lines. The GENIA tagger can assist with this, since it offers named-entity tagging, marking salient biomedical named entities such as proteins, DNA and cell lines as such. By treating named entities as atomic lexical items (effectively words-with-spaces) in the manner described in Section 3.3.6, we were able to boost raw coverage using the ERG over sentences from the biomedical abstracts from 85.9% to 87.5% since it avoided parse failures caused by confusion over the frequently-complex entity names, which

often contain punctuation characters such as hyphens and brackets. However in the first iteration of treebanking (discussed in Section 5.3.1), we found that this NE-tagging was often too zealous, resulting in undesirable analyses. A phrase like *T cells* could be treated as a named entity and lose relatively little information over the two-token noun-compound analysis (although we would argue that the more informative compound analysis is still more valuable and should be used where possible). The problems are clearer with some of the longer strings which the tagger marked as named entities (bracketed here): *[endogenous thyroid hormone and retinoic acid receptors]*, *[octamer-binding transcription factors]* and *more [mature cells]*. In all of these cases there is rich internal structure which the grammar is able to decode (and which could be valuable to a hypothetical downstream system), but which would be made unavailable by this preprocessing (sometimes even ruling out correct analyses, such as when it disrupts co-ordination or adjective phrases as shown above).

The compromise solution was to be as general as possible, to maximise the possibility of a good quality parse being available. Where a multiple-token named entity occurs, we supply two variants to the parser – one with the named entity marked off as an atomic token, and one with the components supplied as if it had not been tagged. PET is able to accept this so-called "lattice" input, and choosing between the different candidate parses corresponding to each version is delegated to the parse selection model. The increased parse coverage and better parse quality made this a worthwhile strategy, with the downside that it expands the number of possible parses, making parse selection more difficult.

For automatic parse selection, this means that there are more trees to rank, and thus more chance the wrong one will be selected. For treebanking, the main effect is an increase in the number of decisions needed. However in addition to this, it is possible that the increased ambiguity will cause the best candidate tree to be ranked outside the top-500 parses that we consider, since the larger parse space means that there are more trees which are able to be ranked higher than the optimal tree. Thus, in some small percentage of trees, we may still be unable to select the correct parse even if it is licensed by the grammar, where that would have been possible without the more general lattice input. In general, however, our anecdotal experience in the treebanking stage was that this configuration was the best one for making the optimal parse available.

One other refinement we added was to hyphenated compounds, which are not split apart by the GENIA tagger (because it follows the PTB tokenisation guidelines for the most part), but are ultimately separated by the ERG. In a phrase such as *mitogen-stimulated incorporation*, the GENIA tagger would usually mark *mitogen-stimulated* correctly as an adjective ('JJ'), but when subsequently split by the grammar, this tag would be duplicated to both *mitogen* and *stimulated*, blocking the correct adjective compound analysis in the ERG, which requires the left member to be a noun. Heuristically assigning the POS-tag 'NN' (i.e. common noun) to leftmost members of

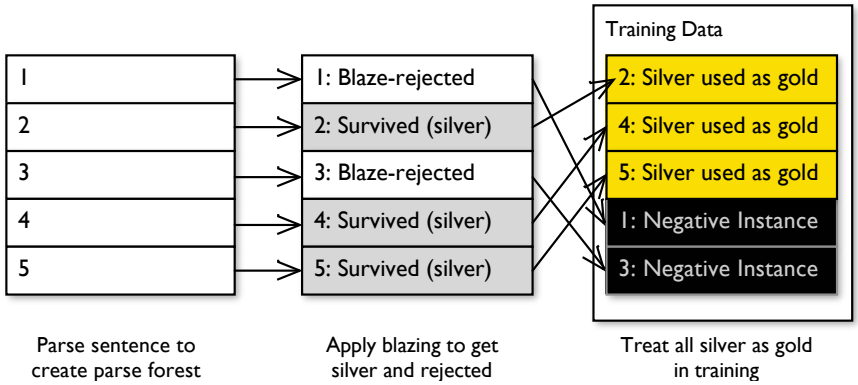


Figure 5.19: Naive strategy for training a model from a blazed parse forest, using a 5-tree parse forest, where we could have up to 500 trees

hyphenated compounds tagged as 'JJ' after splitting greatly improved the handling of such cases, although this approach may not generalise to other domains.

5.4 Parse Selection

Our first set of experiments was designed to evaluate the impact of blazing on parse selection, specifically in a domain-adaptation scenario. As mentioned in Section 3.3.5, parse selection is the process of selecting the top n parses, using a discriminative statistical model trained using the correct and incorrect trees from the treebanking process. However, as explored extensively in Chapter 4, statistical models are highly sensitive to differences in domain, and ideally, we should domain-tune using in-domain treebank data. In this section, we attempt to make maximal use of annotations transferred from the GTB to build a parse selection model for the ERG. We augment the information from the GTB with a strategy related to self-training, as described in more detail below, in order to more fully constrain the parse forests of the corresponding sentences.

In the following sections we describe the particular blazing configurations used for this set of experiments, then explain our evaluation metrics and the various experimental configurations we looked at. Finally we show our results, and discuss some of the other configurations that failed to work, with some of the blazing-related statistics that help explain why.

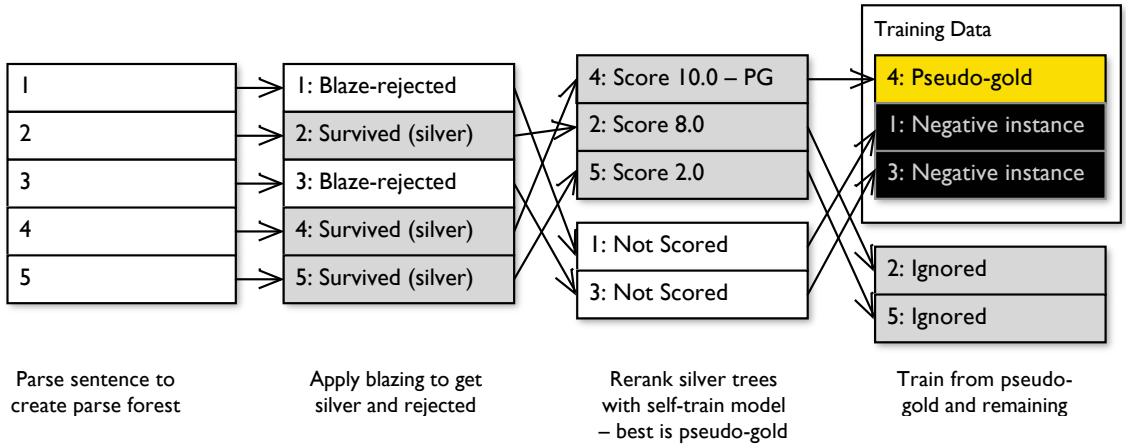


Figure 5.20: The most successful strategy for training from a blazed parse forest, combining blazing with lightweight self-training by only using the top-ranked silver tree as pseudo-gold in training

5.4.1 Experimental Configuration

We create a parse forest by parsing 10747 sentences from a GTB subset not overlapping with the test corpus, using the WeScience model to determine the top 500 parses. As a first attempt to evaluate the efficacy of the blazing procedure, we trained a parse selection model in the most naive way, by treating all silver trees (recall from Section 5.2 that these are those trees which have not been rejected by the blazing process) as though they were gold, as illustrated in Figure 5.19. However, as will become clear below, this method performed poorly, achieving significantly worse parsing results than the benchmark strategy of using WeScience only. The MaxEnt learner TADM also allows non-integer "frequencies" to be assigned to training instances, so we experimented with range of other strategies for assigning these, such as setting the frequency of several top-ranked parses to the reciprocal of their rank within the forest. However, these strategies performed no better than the poorly-performing naive model, and we do not report results for them.

Since a WeScience model was a solid performer over the data, we experimented with using this model to artificially add extra discrimination on top of what was provided by the blazing. The best-performing method we found to create parse selection models from this parse forest was to apply the blazing configurations to determine the silver trees, and then select the top parse from that set, according to the WeScience model. Following the terminology introduced in Section 2.6.1

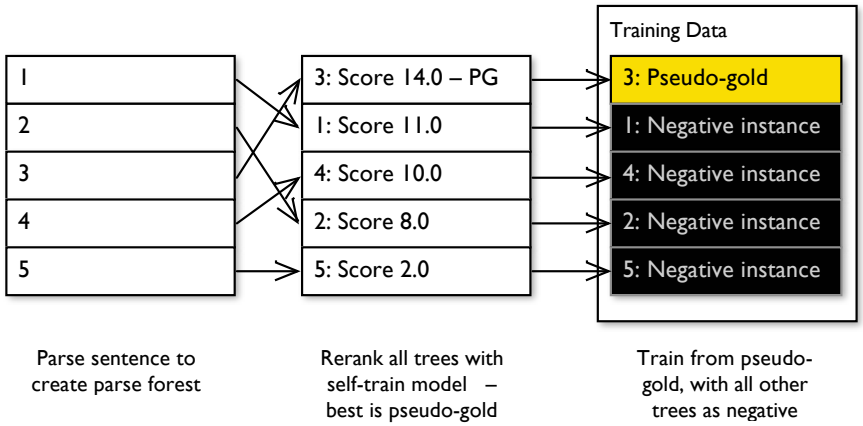


Figure 5.21: A representation of self-training, where the top-ranked tree from some model is used as pseudo-gold – cf Figure 5.20

with reference to self-training, we call this top parse our *pseudo-gold* analysis since the meaning is largely the same, even though it is selected only from the silver trees rather than the entire parse forest. For training the parse selection model, rejected trees are used as negative training data as usual while the remaining silver trees are ignored, since there is a good chance that several are high quality analyses, so we would not wish to use them as negative training data.

For an example of this, if we had 5 possible parses, and blazing removed trees 1 and 3 as candidates, we would rank trees 2, 4 and 5 using a WeScience model. If the top-ranked tree was tree 4, it would become the sole positive training instance (as if it were human-annotated), while trees 1 and 3 would be the negative training instances, and trees 2 and 5 would be ignored. This process is illustrated in Figure 5.20.

This strategy is closely related to self-training (which is described extensively in Section 2.6.1 and explored for the ERG in Section 4.4.7), and differs only in that the pseudo-gold tree comes from some filtered subset (the silver trees) and the other silver trees are not used as negative data. Thus, the question naturally arises of whether the improvements we see are due entirely to this self-training process. Indeed, we showed for different corpora in Section 4.4.7 that self-training was an effective strategy for domain-adaptation, obtaining significant improvements against using solely out-of-domain gold annotated training data. To evaluate the size of the self-training effect here, we ran experiments using the same training sentences reranked with a WeScience model, marking the highest-ranked parse in each case as correct, and all remaining parses in the parse forest of up to 500 trees as incorrect, and then trained

a model with the same set of parameters. This procedure is shown in Figure 5.21 to illustrate graphically how it contrasts with the more informed blazing strategy.

As another point of comparison, we also compare self-training and the blazed self-training combination against a random baseline of selecting a pseudo-gold tree at random from the full 500-tree parse forest, and using the remaining trees as negative data (this is effectively self-training with a random model). We then trained parse selection models using the gold standard out-of-domain (WeScience) data plus the in-domain pseudo-gold analyses from each configuration, and evaluated by parsing our test corpus.

5.4.2 Fallbacks between Strategies

Recall from Section 5.2.1 that the ideal case is where sentences are usably blazed, so that there are both silver trees and rejected trees, meaning that blazing has added information. The other cases are unblazed and overblazed, where blazing allows or rejects the entire parse forest. These cases are the inverse of each other – the entire parse forest is silver or rejected, but from the point of view of parse selection training, they are equivalent in that provide no useful training data. Blazing has added no extra information, so we have no discrimination of the parse forest which could be used to train a model. Extending the partial self-training strategy described above in Section 5.4.1 further, we could in principle use a standard self-training strategy (i.e. the top-ranked tree as pseudo-gold and other trees as negative data) on these sentences, but this is not something we have explored.

Since we have several blazing strategies with different levels of restrictiveness, we also experiment with a fallback configuration. Using this, we apply the first strategy, and if the sentence is not usably blazed (i.e. is either overblazed or unblazed), we fall back to the second strategy, and so on. In the fallback configuration, the blazing strategies are separated by a comma, with the highest-priority strategy listed first. Falling back from a less restrictive to a more restrictive strategy is an example of applying the minimal amount of constraints necessary as discussed in Section 5.2.

5.4.3 Parse Selection Evaluation

We use the same evaluation metrics here as in Chapter 4: exact match as well as dependency triples. For exact tree match, which is again denoted Acc_N for an exact match occurring within the top N trees, we report numbers for Acc_1 and Acc_{10} (following Zhang *et al.* (2007)). For evaluating over dependency triples, we again use EDM, which was described in Section 3.6.3 — specifically, the EDM_{NA} variant, which is most comparable to other work.

GTB Training Config	Gold Added	Acc_1	Acc_{10}	EDM_{NA} P / R / F
N/A (WeSc only)	WeSc	12.3	39.2	82.4 / 79.2 / 80.7
Random	WeSc	6.1	20.0	70.7 / 70.2 / 70.5
IgnoreEqPar (Full)	–	2.4	16.1	71.4 / 71.2 / 71.3
IgnoreEqPar (Full)	WeSc	3.1	17.9	73.1 / 71.2 / 72.1
BNC +RP +MP (Full)	WeSc	6.1	19.4	75.5 / 69.8 / 72.5
IgnoreEqPar (Self-train)	–	12.0	33.3	82.6 / 79.6 / 81.1
Self-training only	WeSc	12.9	39.2	82.4 / 80.3 / 81.3*
IgnoreEqPar (Self-train)	WeSc	12.9	39.2	83.5 / 80.9 / 82.2***††
[**BNC +RP**], **RP** (Self-train)	WeSc	13.3	39.9	84.0 / 80.5 / 82.2***†
RP, **IEP** (Self-train)	WeSc	13.0	39.6	83.7 / 80.9 / 82.2***††
BNC +RP (Self-train)	WeSc	13.3	39.9	84.1 / 80.8 / 82.4***††
RaisePremods (Self-train)	WeSc	13.3	40.1	83.8 / 81.2 / 82.5***†††

Table 5.5: Results over the ERGGTB development corpus. "WeSc only" shows parsing using a pure WeScience model. Other configurations used models trained by automatically restricting the same GTB training sentence parse forest, setting a pseudo-gold tree either randomly (single tree), using self-training (single best from a WeScience model), using "full" blazing (all silver trees as gold) or "self-trained" blazing (single highest-ranked of the silver trees as pseudo-gold, other silver trees discarded) in two different configurations. The gold WeScience data is also used for training in all but two cases (as shown). Significance figures are against "WeSc only", (*: $p < 0.05$; ***:$p < 0.001$), and "Self-train", ($\dagger\dagger$: $p < 0.01$; $\dagger\dagger\dagger$: $p < 0.001$)

5.4.4 Parse Selection Results

We present our results in Table 5.5, including the best-performing blazing configurations, the self-training results and the weak baseline trained on a random tree from the same GTB parse forest as used in blazing. Not shown there is the estimate for exact match accuracy for a random baseline to give an idea of the difficulty of parsing the corpus. For Acc_N if we have p parses for a sentences, the probability of randomly selecting one correctly is $\min(1, \frac{N}{p})$. Averaging over sentences, we get a random baseline for our test corpus of 1.1% / 6.8% for Acc_1/ Acc_{10}. This baseline is difficult to calculate rigorously for EDM due to the difficulty of parsing truly randomly with PET.

We also show the parsing accuracy results obtained using only out-of-domain data, designated "WeSc only", as a strong baseline. We see some evidence that self-training can be a useful domain-adaptation strategy, giving a weakly significant F-score im-

	IEP	**RP +MP**	**RP**	**RP +BNC**	**RP +BNC +MP**
Discrims/Sent	144.2	144.2	144.2	144.2	144.2
Rejected/Sent	40.8	42.5	42.8	44.2	45.7
Unblazed Sents	3.9%	2.5%	3.4%	2.9%	2.1%
Overblazed Sents	14.2%	20.7%	15.3%	15.6%	24.6%
Usably Blazed Sents	81.9%	76.8%	81.3%	81.5%	73.3%
Trees/Sent (overall)	423.3	423.3	423.3	423.3	423.3
Silver/Sent (blazed)	98.4	63.1	88.5	74.9	48.5
Silver/Sent (usable)	120.1	82.2	108.8	91.9	66.2

Table 5.6: Blazing Statistics, over all 10736 parseable training sentences. The first block shows discriminants available per sentence, and how many were rejected by the blazing (which removes at least one tree). The second block shows percentage of unblazed sentences (no discriminants rejected), overblazed sentences (all trees removed) and usably-blazed sentences (at least one removed and one silver tree remaining). The third block shows how many parses were produced initially, the average number of silver trees remaining over blazed sentences (including overblazed with 0 trees) and the average number remaining over usably blazed sentences.

provement over using WeScience only. This echoes the results for self-training targeting different test corpora in Section 4.4.7, although the EDM improvement is more modest here, with only 0.6% instead of 1.3–1.4% which we saw in Table 4.11. More importantly, our blazing strategy yields strongly significant F-score improvements over both the strong baseline out-of-domain model and the standard self-training.

5.4.5 Parse Selection Discussion

There is strong evidence that these blazing methods can help create a parse selection model to give a significant boost in dependency score without requiring any additional human annotation. The exact match results are inconclusive – the best improvement we see is not significant, although the granular EDM metric may provide a better reflection of performance for downstream applications in any case.

In our initial investigations, a range of configurations failed to provide improvements over the baseline. If we do not augment the training data with human-annotated WeScience data, the performance drops. Also, if we treat all silver trees as pseudo-gold instead of using the self-training/blazing combination as described,

the model performs very poorly. Table 5.6 provides some explanation for this. Over sentences which provide usable discriminative training data (at least one incorrect and one silver tree), if we apply **IgnoreEqPar**, 120 silver trees remain, so it is failing to disambiguate sufficiently between the ERG analyses. This is like to be partially due to an imperfect transfer process, and partially due to shallower, less precise GTB analyses.

Naively we might have hoped that accounting for some obvious cases where the GTB is insufficiently restrictive (such as by adding bracketing to noun compounds) would eliminate most of the ambiguity (without making assumptions about the accuracy). However, after observing that the less restrictive strategies were not constraining the parse forest enough, we evaluated the more restrictive **IEP + MP + BNC** configuration. This does attempt to add structure to noun compounds (possibly too much, as we pointed out in Section 5.2.2), but there are still 66 silver trees per usable sentence on average, so a large amount of ambiguity remains. More interestingly, we can see from the 'Full' results in Table 5.5 that there is fairly little corresponding performance improvement. One factor affecting this could be that the training set size (i.e. the number of usably blazed trees) is effectively 10% smaller, but it still seems clear that more reduction in ambiguity is necessary. However the partial self-training method we supplied seems to be a relatively effective means of achieving this.

There are cases of ambiguity we could not hope to resolve given the less precise nature of the GENIA treebank – for example, it is not possible to recover distinctions between complements and adjuncts in most cases. The ambiguity discussed in the example Section 5.2.3 about whether the *by*-PP denoted a passive agent was in fact a special case of complement/adjunct ambiguity.

The extreme case of ambiguity is of course the unblazed sentences, where the blazing configuration is unable to suggest any discriminants to rule out. We can see from Table 5.6 that these are a fairly small percentage of the total, so do not correspond to a huge amount of training data. The other extreme, of overblazed sentences, may superficially appear more concerning, since this means we are losing more than 14% of training sentences, and up to 25% in some configurations. However, it is likely that some of these cases of overblazing correspond to instances where there is no correct parse within the forest, meaning that no valid training data can be obtained from the sentence. That is, blazing could be making the same decision that a human annotator would, by rejecting all analyses, which is in some cases the right decision. Indeed, for the gold standard treebank we use, the human annotators rejected all candidate trees for 13% of sentences, and while this would not align completely with those rejected by blazing, some evidence shows that there is substantial overlap. If we apply the **RaisePremods** blazing strategy to the ERGGTB sentences, the overblazed sentences have precision/recall/F-score of 57.5/61.1/59.2% against the sentences which were manually rejected by humans. This suggests that in more than half of all cases, it was appropriate for blazing to not suggest any good quality sentences, as such sentences did not exist in the data according to human judgement in any case.

While we can alleviate the problems from the more restrictive strategies by using the fallback procedure described in Section 5.4.2, this can only reduce the number of overblazed sentences to those in the least restrictive strategy. If we did wish to avoid overblazing, we could attempt to relax the criteria for choosing candidate constituents to examine for conflicts. However, it is not clear how this could be relaxed any further than in **IgnoreEqPar** in a principled way. It might be possible to exclude either small or large constituents in an effort to get extra training data out of the 14% of overblazed sentences (or at least the subset that actually corresponds to sentences with good parses available). In any case, the results above suggest that there may not be much corresponding reduction in ambiguity, unless there is some special characteristic of these sentences which means that the parse forests are easy to constrain, which could have contributed to their being overblazed in the first place.

Overall, the strategy of overlaying blazing with self-training has provided useful performance improvements, although it is possible that more optimisations to the process are possible.

5.5 Reducing Treebanking Labour

Blazing is designed to reduce the size of the parse forest, so it seems natural to evaluate its impact on the treebanking process, and whether we can reduce the amount of time and number of decisions required to enable more efficient treebanking. Indeed, as mentioned above, Tanaka *et al.* (2005), performed an analogous procedure using POS labels, so it seems natural to investigate whether having more detailed syntactic annotations available can also be useful.

5.5.1 Selecting a Blazing Strategy

There are a range of blazing strategies, some of which can be used in combination, which have varying levels of validity and restrictiveness. The strategies we use here are a subset of those described in Section 5.2.2 and summarised in Table 5.1, which were used in the parse selection experiments.

Ideally, during the treebanking process we would start with a more restrictive blazing strategy, and have the ability to dynamically fall back to a less restrictive strategy, but this capability is not present in [incr tsdb()]. The approach we used is based on number of decisions being logarithmic in the number of trees. If we can get around 40 or fewer silver trees, the remaining treebanking is very tractable and fast, as we only require a few decisions chosen from just a handful of remaining discriminants. However further restriction below this saves relatively little time, and increases the chance of the blazing removing the correct tree, which we wish to avoid, since [incr tsdb()] does not allow the treebanker to modify either manual or blazed decisions, except by clearing and starting from scratch.

		Standard		Blazed	
		Mean	Med	Mean	Med
Ann 1	Manual Decisions	6.25	7	3.51	4
	Automatic decisions from blazing	0.00	0	37.60	32
	Reverted automatic decisions	0.00	0	10.21	0
	Time (sec)	150	144	113	107
Ann 2	Decisions	6.42	7	4.68	4
	Automatic decisions from blazing	0.00	0	34.55	27
	Reverted automatic decisions	0.00	0	10.69	0
	Time (sec)	105	101	96	80

Table 5.7: Comparison of number of decisions and treebanking time (mean then median) using the fallback blazing configuration discussed in the text (least restrictive strategy which gives less than 40 trees) over 80 sentences for each column. Also shown are the number of decisions made automatically in advance by the blazing module, and of those how many were subsequently reverted manually by an annotator attempting to create the best parse tree

The parse forest for treebanking was created by having a list of candidate blazing strategies using various combinations of the strategies listed in table 5.1, ranging from the least restrictive **IgnoreEqPar** to the most restrictive **RaisePremods +BracketNCs +MapPOS** combination. For each sentence, we select the least restrictive strategy which still gives us fewer remaining trees than a threshold of 40. If no strategies do so, we use the strategy which gives us the fewest trees for the given sentence. This is using the principle of applying the minimum constraints necessary for our target application. Here, since our target application is post-filtering by a human treebanker, reducing the parse forest to 40 silver trees is considered sufficiently restrictive. Using another subset of the GENIA treebank containing 864 parseable sentences, 48% of sentences came below the the threshold of 40, while 36% were above and 16% were not disambiguated at all. Most sentences (41%) used **IgnoreEqPar** alone, the least restrictive configuration.

5.5.2 Blazed Treebanking Results

For this strategy to be useful for treebanking, it should be both more efficient, in terms of fewer decisions and less annotation time, and valid, in terms of not introducing a bias when compared to conventional unblazed treebanking. To evaluate these questions, we selected 160 sentences at random from the previously described parse forest of 864 sentences. These sentences were divided randomly into four equal-sized

	Ann. 1			
	Std	Blz		
Agreed Sentences	42.5	45.0	Std	Ann. 2
Agreed, excl rej	32.4	33.3		
Rejection F-score	80.0	82.4		
Constituent F-score	88.7	87.6		
Agreed Sentences	42.5	57.5	Blz	
Agreed, excl rej	39.5	45.2		
Rejection F-score	44.4	78.3		
Constituent F-score	86.2	84.8		

Table 5.8: Agreement figures for different combinations of blazed and unblazed overlap between annotators 1 and 2. Each cell corresponds to 40 sentences. 'Agreed' is the percentage of those with an identical tree selected, or all trees rejected. 'Agreed, excl rej' ignores sentences rejected by either annotator. 'Constituent F-score' (also excludes rejections), gives the harmonic mean of the labelled per-constituent precision. 'Rejection F-score' is the harmonic mean of the precision of rejection decisions

groups: blazed for both annotators, standard for both annotators, and two groups blazed for one annotator only, so we could compare data about timing and decisions between the standard and blazed sentences for each annotator, and inter-annotator agreement for each possible combination of blazed and standard treebanking. The divisions took no account of whether we were able to usably blaze the sentences, reflecting the real-world scenario, so some sentences in the blazed configuration had no restrictions applied. The items were presented to the annotators so they could not tell whether the other annotator was treebanking in standard or blazed configuration, to prevent subconscious biases affecting inter-annotator agreement. The experiments were conducted after both annotators had already familiarised themselves with the treebanking environment as well as the characteristics of the domain and the annotation guidelines.

Annotators worked in a distraction-free environment so we could get accurate timing figures. The treebanking GUI records how many decisions were made as well as annotation time, both important factors in annotation efficiency. The results for efficiency are shown in Table 5.7 where we see a 43% reduction in the mean decisions required for annotator 1, and 27% reduction for annotator 2. Annotator 1 also shows a substantial 25% reduction in mean annotation time, but the time decrease for annotator 2 is only 8%. In 30% of successfully-blazed sentences, the annotators cleared all blazed decisions, suggesting it is sometimes too zealous.

For agreement, we show results for the strictest possible criterion of exact tree match. For a less blunt metric that still roughly reflects agreement, we also follow Tanaka *et al.* (2005) in reporting the (micro-averaged) harmonic mean of precision across labelled constituents indexed by character span, where constituents selected by both annotators are treated as gold (inaccurately denoted 'F-score' for brevity). Annotators should also agree on rejected trees, where no parses are valid. In Table 5.8, we show exact match agreement accuracy (identical trees and matching rejections both count as correct), as well as the same figure ignoring sentences rejected by either, and the harmonic mean of precision of both labelled constituents and tree rejections. The agreement figures are similar between cells, with the notable exceptions being higher exact match when both annotators had blazed forests, and a surprising dip in the rejection "F-score" in the bottom left cell.

5.5.3 Blazed Treebanking Discussion

The reductions in mean numbers of decisions strongly support the efficacy of this technique, although the discrepancies between the annotators suggest that the different treebanking techniques may be more or less amenable to speed-up using these tools. The timing figures are somewhat more equivocal, although still a substantial 25% for annotator 1 (the author of this thesis). This is partially to be expected, since some of the treebanking will be taken up with unavoidable tasks such as evaluating whether the final tree is acceptable that blazing cannot avoid. However, the 8% reduction in mean annotation time for annotator 2 (Dan Flickinger) is still fairly modest. This could be affected by annotator 2's more extensive treebanking experience — as the primary architect of the ERG, he has treebanked many thousands of sentences for the LOGON and WeScience corpora, among others. This could lead to a lower baseline time, with less room for improvement, but as we still see a 21% reduction in median parsing time, another important effect could be a few outlier sentences inflating the mean for the blazed configuration.

For agreement, we are primarily concerned here with whether blazing introduces a bias that is distinguishable from what we see when annotators are working under standard non-blazed conditions — which may be manifested in decreased agreement between configurations where only one annotator has blazed data, and when both have non-blazed data. Thus the fact that we see quite similar agreement figures between the half-blazed and standard configurations is very encouraging (apart from the low F-score for rejections in one cell). This small amount of data suggests that any changes in the resultant trees introduced by blazing are hard to distinguish from the inevitable "background noise". Given this, the fact that we see a noticeably higher exact match score when both annotators have blazed sentences suggests we may be justified in using blazing to improve inter-annotator agreement, although the lower constituent score may indicate we have insufficient data to reach that conclusion.

5.6 Summary

In this chapter, we investigated the task of transferring annotations between incompatible formalisms, as applied to the ERG. The procedure we use is denoted *treeblazing*, and involves using annotations from an external phrase structure treebank to constrain the parse forest produced by a precision HPSG grammar. To exemplify the principles involved, we used the GENIA treebank for source annotations and the ERG as the target grammar, with the WeScience corpus as a secondary source of out-of-domain training data. However the methods we investigated would in principle be applicable to any similar phrase structure treebank and similar HPSG grammar.

The source trees were mapped onto corresponding ERG parse forests and used to exclude incompatible trees. There were a range of strategies that were used for the mapping to handle differences in conventions between the ERG and the GTB. These included targeted ignoring of particular nodes which are known to cause spurious conflicts, and systematic remapping of sections of the parse tree to ensure a closer match. We found that the resulting partially-constrained parse forests were useful for two purposes. For building a parse selection model, we combined the treeblazing technique with a strategy inspired by self-training approaches that have been successful elsewhere (including for the ERG, in Chapter 4), where the best tree according to some parse selection model is selected from the partially-constrained forest, rather than following the standard practice in self-training of selecting it from the full parse forest. This enabled the creation of a parse selection model for the ERG which was better adapted to the domain of biomedical abstracts without requiring additional human annotation effort, improving accuracy significantly over using an out-of-domain model and even over using self-training.

We also investigated using the treeblazing process as a prefilter to improve treebanking efficiency. This substantially reduced the annotation time required per tree, making the annotation of gold-standard in-domain data more tractable and efficient, which could be useful for creating a larger test corpus or additional training data. Additional training data could be used to create even better domain-adapted parse selection models than those achievable with the fully-automatic approach of using treeblazing only, so the choice between the two would depend on the availability of treebanking expertise and funding, but there is clearly value in either strategy.

In this chapter and the previous chapter, we explored various possibilities for producing better parse selection models for deep parsing in new domains. In the next part of the thesis, we will look at ways we might use the outputs of a deep parser, in some cases making use of these better parse selection models.

Part III

Applications of Deep Parsing

Introduction

In Part II, we investigated techniques to improve parsing accuracy using precision grammars, and in particular the ERG. We evaluated the question of how to maximise parsing accuracy given some domain and some set of training corpora, as well as techniques to assist with the creation of those corpora. In other words, we examined ways to improve deep parsing.

In this part, we look at the complementary question of how we can use the outputs of a deep parser. Improving parsing performance would be of little use if we had no downstream task for which we wanted to use the outputs.

We examine two tasks as an investigation of how we can apply the ERG in these cases. Firstly we look at a well-defined task in information extraction over biomedical research abstracts, and how the MRS outputs produced by the ERG can assist with this. Secondly, we consider the question of matching DMRSs to each other in a hypothetical semantic retrieval system, and how to abstract away from less important distinctions in this scenario.

Chapter 6

Biomedical Information Extraction

6.1 Introduction

Our first application of precision parsing is on a rigidly defined shared task in biomedical language processing. Specifically, we present a system for Task 3 of the BioNLP 2009 Shared Task (BN09ST: Kim *et al.* (2009)); we gave a high-level overview of the shared task description in Section 2.8.2.

Briefly, the shared task concerns the detection of biomedical events and the biomedical entities which participate in them, primarily to support the creation of curated databases. Task 3 requires detecting modification of these events — that is, determining whether they are subject to speculation or negation. As a reminder of how Task 3 works, we have reproduced the example from Section 2.8.2. In (6.1) we show the same sentence, displaying the protein annotations (provided as part of the source data set) and the event triggers (which are needed for Task 1).

(6.1) $[_{\text{protein}}$ *TRADD* $]_1$ *was the only protein that* $[_{\text{trigger}}$ ***interacted*** $]_4$ *with wild-type* $[_{\text{protein}}$ *TES2* $]_2$ *and not with isoleucine-mutated* $[_{\text{protein}}$ *TES2* $]_3$.

Task 1 also requires event argument annotation; the arguments for one of the events in the sentence are shown in (6.2).

(6.2) Event evt_2
TYPE = BINDING
TRIGGER = $[_{\text{trigger}}$ *interacted* $]_4$
THEME_1 = $[_{\text{protein}}$ *TRADD* $]_1$
THEME_2 = $[_{\text{protein}}$ *TES2* $]_3$

For Task 3, the primary focus of this chapter, participants were required to detect modification of such events in the form of SPECULATION or NEGATION. In this example, there is an instance of NEGATION for Event evt_2, since according to the sentence *TRADD* did not interact with *isoleucine-mutated TES2*. We represent this in modification instance in (6.3).

Event Type	**Argument(s)**
Gene Expression	Theme(Protein)
Transcription	Theme(Protein)
Protein Catabolism	Theme(Protein)
Phosphorylation	Theme(Protein)
Localization	Theme(Protein)
Binding	One or more Themes(Proteins)
Regulation	Theme(Event/Protein), Cause(Event/Protein)
Positive Regulation	Theme(Event/Protein), Cause(Event/Protein)
Negative Regulation	Theme(Event/Protein), Cause(Event/Protein)

Table 6.1: Event Types and Their Arguments

(6.3) Modification mod_1
type = Negation
theme = evt_2

These modification events often depend on long-distance syntactic relationships. The example shown above is relatively straightforward and short for clarity, but even in this simple case, the syntactic knowledge required to recognise the negation is fairly sophisticated. We must recognise that *isoleucine-mutated TES2* is the complement of the preposition *with*, that the whole prepositional phrase modifies the event trigger verb *interacted*, and also that the negation particle *not* negates the semantics of the entire phrase. There is likely to be value from utilising a sophisticated syntactico-semantic analysis to determine such negation instances, and deep parsing could provide such an analysis. Thus, Task 3 seemed a good candidate for applying deep parsing techniques, and that is the focus of this chapter.

We investigate the application of a machine-learning approach using features derived from the semantic output of ERG parses of the shared task sentences. We also evaluate the effects of different parse selection models for producing the ERG parses on the performance in this downstream application.

6.2 Task Description

Here we expand on the description of the shared task in Section 2.8.2. We noted that Task 1 was concerned with identifying biomedical events involving the provided protein annotations, and classifying these events into one of eight types. These types are all selected from the GENIA ontology (Kim *et al.* 2006) of biomedical terminology,

to which the reader is referred for a more detailed description of event types. The event types are related, unsurprisingly, to protein biology, and are shown in Table 6.1

As noted in the table, all event types require at least one THEME argument. The first three event types concern the creation and breakdown of proteins, while PHOSPHORYLATION refers to a kind of protein modification. These are the only event types which are specific to proteins. The remainder of the events have more general denotation in the GENIA ontology, but always involve proteins in the instances annotated in the provided data. These four event types can be defined as *simple events*, since they involve exactly one protein as THEME. The final simple event type is LOCALIZATION, a fundamental molecular event referring to the transport of cellular entities.

BINDING refers to a particular kind of interaction of a molecule with sites on another molecule. It thus generally involves multiple entities, so events of this type can have more than one protein theme. REGULATION refers to a process which affects the rate of another process. POSITIVE REGULATION refers to increasing the rate of this other process while NEGATIVE REGULATION refers to reducing it; both of these can be considered subtypes of the more general REGULATION, which is used when it is not clear whether the regulation is positive or negative. REGULATION event types all have the same, slightly more complex structure. There is a THEME argument which can be either a protein entity or, unlike the other event types, another biomedical event entity, as it is possible for events themselves to be subjected to regulation. There is also a CAUSE argument, which is also a protein or event, and denotes the agent which caused the regulation to occur.

The annotation guidelines require that all events be linguistically grounded, so all events must be associated with a *trigger word*, which is the textual hint that the event occurred. The primary evaluation metric is "approximate recursive matching", described in detail in Kim *et al.* (2009).

6.3 Event Modification Detection for Task 3

For the purposes of this chapter, as in MacKinlay *et al.* (2011b), we treat Task 1 as a black box, and base our event modification classifiers on the output of a range of Task 1 systems from the original BioNLP 2009 shared task, namely: the best-performing Task 1 system of UTurku (Björne *et al.* 2009), the second-best Task 1 system of JULIELab (Buyko *et al.* 2009), and the mid-ranking Task 1 system of NICTA (MacKinlay *et al.* 2009). For the majority of our experiments, we use the output of UTurku exclusively.

For Task 3, we ran a syntactic parser over the abstracts and used the outputs to construct feature vector inputs to a machine learner. We built two separate classifiers for each training run: one to identify SPECULATION and one for NEGATION. A two-classifier decomposition (rather than, say, a single four-class classifier) seems the most natural fit for the task. SPECULATION and NEGATION are independent of one

another (informally, but not necessarily statistically) and it enables us to focus on feature engineering for each subtask, although we can of course use some of the same features for each classifier when we expect them to be beneficial in each case.

6.3.1 Deep Parsing with the ERG

Intuitively, it seemed likely that syntactico-semantic analysis would be useful for Task 3. To identify NEGATION or SPECULATION with relatively high precision, it is probable that knowledge of the relationships of possibly distant elements (such as the NEGATION particle *not*) to a particular target word would provide valuable information for classification.

Further to this, it was our intention, in line with the rest of this thesis, to evaluate the utility of deep parsing in such an approach, rather than a shallower annotation such as the output of a dependency parser. We briefly argue for the plausibility of this notion alongside a concrete example in Section 6.3.3. The primary source of data for classification that we use is, therefore, the ERG. As we saw in Chapter 4 it is relatively robust across different domains. However, it is a general-purpose resource, and there are some aspects of the language used in the biomedical abstracts that cause difficulties.

We have already discussed some techniques to handle these in Section 5.3.2. The handling we use for most of this chapter differs slightly from these, primarily because the majority of the work in this chapter was conducted at an earlier stage of the research in this thesis, although the general principles are similar. Again, we use the GENIA tagger to supply POS-tags for use in handling unknown words (i.e. those missing from the ERG lexicon). In this work, however, we do not defer to TnT for tagging nominal elements. We also use named entity tagging for this chapter, but in this case we do not supply alternate input lattices for decomposed and atomic versions of the named entities — we simply supply the atomic version only. We return to the question of preprocessing and compare these methods with the newer techniques in Section 6.6.2.

For the first part of this chapter, we also follow MacKinlay *et al.* (2011b) in using the '0902' version of the ERG (which is substantially older than the versions used in Chapters 4 and 5) as this was the newest release of the grammar when this research was originally undertaken. This release added support for *chart-mapping* functionality (Adolphs *et al.* 2008) over the earlier versions, allowing various optimisations to its handling of tokenisation and unknown words, improving the parse coverage over using earlier versions of the grammar such as the older grammar used in MacKinlay *et al.* (2009). With the 0902 grammar the coverage is 76% over the training corpus. However this still leaves 24% of the sentences inaccessible, so a fallback strategy is necessary. Some methods by which we achieve this are discussed below. In Section 6.6.1, we will return to the question of selecting a grammar version and examine the impact this has on the system.

6.3.2 Feature Extraction from RMRSs

We have previously discussed how the ERG is designed to produce semantic output in the form of MRS, or the variant Robust MRS (RMRS), which we described briefly in Section 3.5.2, and which is intended to be more easily produceable by processing components of shallow and intermediate depth. RMRS is used to create all of our parser-derived features.[1] We show another concrete example of an RMRS in Figure 6.1, and highlight some aspects which are important for feature extraction, repeating points which are particularly useful in this chapter.

An RMRS is, as we have previously mentioned, a bag of *elementary predicates*, or EPs. Each EP shown is associated with:

1. A predicate name, such as `_require_v_1` (where 'v' indicates the part-of-speech is verb).
2. Character indices to the sources sentence, such as `<98:106>`.
3. A label, in the form of a handle such as `h20`, listed as the first item in the parentheses following the predicate. Equality of labels indicates that the EPs themselves are related
4. An `ARG0` argument, as the second item in parentheses, such as `e2:`, generally referring to the variable introduced by the predicate.
5. A number of other arguments, which are listed separately to the main EP in RMRS.

The primary differences with the original example from Figure 3.5 are that we have omitted the entity properties, which are not used for the feature extraction in this chapter, and added in character indexes, which are fairly important for feature extraction here.

In constructing features, we make use of:

- The *outscopes* relationship (specifically qeq-outscopes) – if EP A has a handle argument which qeq-outscopes the label of EP B, A is said to immediately outscope B; *outscopes* is the transitive closure of this.

[1] It may seem that the Dependency MRS formalism described in Section 3.5.3 would be a better fit for such an information extraction task. When we ran the original experiments on which this chapter was based, the tools available for DMRS conversion were relatively immature, so DMRS was not a viable option. In a subsequent experiment, we re-implemented the feature extraction algorithms of this chapter using DMRS rather than RMRS, and found that it was possible to achieve the same results with far fewer lines of code, showing that many of the features implicitly use dependency-like structures in any case. However the original work was performed using the logically-equivalent RMRS representation, and there are still certain cases where it is possible to produce RMRSs but not DMRSs; converting from RASP, which we discuss in Section 6.3.5, is one such area. We therefore exclusively use RMRS for the remainder of this chapter.

```
h1
_thus_a_1<62:67> (h3, e5:)
  ARG1(h3, h4:)
  qeq(h4:,h17)
nfkappa b_nn<68:78> (h16, x11:)
udef_q_rel<68:89> (h6, x9:)
  RSTR(h6, h8:)
  BODY(h6, h7:)
  qeq(h8:,h10)
compound_rel<68:89> (h10, e12:)
  ARG1(h10, x9:)
  ARG2(h10, x11:)
udef_q_rel<68:89> (h13, x11:)
  RSTR(h13, h15:)
  BODY(h13, h14:)
  qeq(h15:,h16)
_activation_n_1<79:89> (h10, x9:)
neg_rel<94:97> (h17, e19:)
  ARG1(h17, h18:)
  qeq(h18:,h20)
_require_v_1<98:106> (h20, e2:)
  ARG1(h20, u21:)
  ARG2(h20, x9:)
parg_d_rel<98:106> (h20, e22:)
  ARG1(h20, e2:)
  ARG2(h20, x9:)
_for_p<107:110> (h20, e24:)
  ARG1(h20, e2:)
  ARG2(h20, x23:)
neuroblastoma cell_nn<111:129> (h34, x29:)
udef_q_rel<111:146> (h25, x23:)
  RSTR(h25, h27:)
  BODY(h25, h26:)
  qeq(h27:,h28)
compound_rel<111:146> (h28, e30:)
  ARG1(h28, x23:)
  ARG2(h28, x29:)
udef_q_rel<111:146> (h31, x29:)
  RSTR(h31, h33:)
  BODY(h31, h32:)
  qeq(h33:,h44)
_differentiation_n_of<130:146> (h28, x23:)
  ARG1(h28, u35:)
```

Figure 6.1: RMRS representation of the sentence *Thus NF-kappa B activation is not required for neuroblastoma cell differentiation* showing, in order, elementary predicates along with arguments and qeq-constraints for each EP, which have been indented and placed below the relevant EP for readability.

- The *shared-argument* relationship, where EPs C and D refer to the same variable in one or more of their argument positions. We also in some cases make further restrictions on the types of arguments (`ARG0`, `RSTR`, etc) that may be shared on either end of the relationship.

6.3.3 Comparing Dependency Representation with RMRS

Digressing briefly, we noted in Section 6.3.1 that we were motivated to use deep parsing due to being able to natively handle relationships between distant elements of the sentence. The notion of scoping, and the fact that MRS and its variants natively abstract away from details of constructions such as the passive voice means that particular features are easier to extract than it would be in, for example, a dependency-based representation.

In Figure 6.1, the EP `_activation_n_1` shares an argument with the verb `_require_v_1` as its object `ARG2`. The verb is in turn outscoped by the negation particle `neg_rel` (corresponding to *not*). We can thus trivially see that the *activation* is the target of some form of negation (in a way which it would not be as the object of the verb). It would likely be possible to draw such a conclusion from a shallower dependency-based representation such as the output of the GDep parser (Miyao and Tsujii 2008) which was supplied with the shared task. However it would require extensive post-processing to abstract away from the syntax. Passives are generally not explicitly marked, so we may need to perform some lightweight *semantic role labelling* (Gildea and Jurafsky 2002) which could assign roles such as *agent* and *patient* to work around this problem. In addition, for a complete treatment of similar cases we would probably wish to handle multi-token negation particles such as *not yet*, which are simple extensions when using the MRS representation.

Whether this extra abstraction justifies the extra overhead of parsing using a precision grammar compared to the shallower approach is of course a separate question. The shared task is designed to motivate the development of supporting tools for semi-automatic curation of databases (Kim *et al.* 2009). In such a setting, the deep parsing overhead is unlikely to be particularly significant compared to the manual annotation which will inevitably be required.

6.3.4 Feature Sets and Classification

Feature vectors for a given event are constructed on the basis of the trigger word for the particular event, which we assume has already been identified. We use the term *trigger EPs* to describe the EP(s) which correspond to that trigger word – i.e. those whose character span encompasses the trigger word. We have a potentially large set of related EPs (with the kinds of relationships described above), which we filter to create the various feature sets, as outlined below.

We have several feature sets targeted at identifying NEGATION:

- NegOutscope2: If any EPs in the RMRS have predicate names in {`_no_q`, `_but+not_c`, `_nor_c`, `_only_a`, `_never_a`, `_not+as+yet_a`, `_not+yet_a`, `_unable_a`, `neg_rel`}, and that EP outscopes a trigger EP, set a general feature as well as a specific one for the particle.
- NegVerbOutscope: If any EPs in the RMRS have predicate names in {`_fail_v`, `_cease_v`, `_stop_v`}, and that EP outscopes a trigger EP, set a general feature as well as a specific one for the particle.
- NegConjIndex: If any EPs in the RMRS have predicate names in {`_not_c`, `_but+not_c`, `_nor_c`}, and the `R-INDEX` (right hand side of a conjunction) of that EP is the `ARG0` of a trigger EP, set a general feature as well as a specific one for the particle — capturing the notion that these conjunctions are semantically negative for the particle on the right. This also had a corresponding feature for the `L-INDEX` of `_nor_c`, corresponding to the left hand side of the *neither...nor* construction.
- Arg0NegOutscopeeSA: For any EPs which have an argument that matches the `ARG0` of a trigger EP, if they are outscoped by an EP whose predicate name is in the list { `_only_a`, `_never_a`, `_not+as+yet_a`, `_not+yet_a`, `_unable_a`, `neg_rel`}, set a general feature to true, as well as features for the name of the outscoping and outscoped EPs. This is designed to catch trigger EPs which are nouns, where the verb of which they are subject or object (or indeed an adjective/preposition to which they are linked) is semantically negated.

And several targeted at identifying Speculation:

- SpecVObj2: if a verb is a member of the set {`_investigate`, `_study`, `_examine`, `_test`, `_evaluate`, `_observe`} and its `ARG2` (which corresponds to the verb object) is the `ARG0` of a trigger EP. This has a general feature for if any of the verbs match, and a feature which is specific to each verb in the target list.
- SpecVObj2+WN: as above, but augment the list of seed verbs with a list of WordNet sisters (i.e. any lemmas from any synsets for the verb), and add a feature which is set for the seed verbs which gave rise to other sister verbs.
- ModalOutscope: modal verbs (*can*, *should*, etc) may be strong indicators of Speculation; this sets a value when the trigger EP is outscoped by any predicate corresponding to a modal, both as a general feature and a specific feature for the particular modal.
- AnalysisSA: the `ARG0` of the trigger EP is also an argument of an EP with the predicate name `_analysis_n`. Such constructions involving the word *analysis* are relatively frequent in speculative events in the data.

And some general features, aiming to see if the learning algorithm could pick up other patterns we had missed:

- TrigPredProps: Set a feature value for the predicate name of each trigger EP, as well as the POS of each trigger EP.
- TrigOutscopes: Set a feature value for the predicate name and POS of each EP that is outscoped by the trigger EP.
- ModAdj: Set a feature value for any EPs which have an `ARG1` which matches the `ARG0` of the trigger EP if their POS is marked as adjective or adverb.
- +Conj: This is actually a variant on the feature extraction method, which attempts to abstract away the effect of conjunctions. If the trigger EP is a member of a conjunction (i.e. shares an `ARG0` with the `L-INDEX` or `R-INDEX` of a conjunction), also treat the EPs which are conjunction parents (and their conjunctive parents if they exist) as trigger EPs in the feature construction.

In Section 6.4, we present some experiments exploring the various combinations of RMRS-derived features, which we used to settle on one feature set per modification type for the remainder of chapter.

6.3.5 Extending Coverage using RMRSs from RASP

While the coverage we were able to obtain from the ERG was respectable, it is clearly unwise to have no chance at correctly classifying the events in the remaining 24% of the sentences that we couldn't parse. There are a number of strategies we could pursue to deal with this. One obvious approach might be to augment it with a more simplistic bag-of-words approach, and this is indeed something we investigate, as discussed in Section 6.3.6. However in line with our intuition and experimental evidence that syntactico-semantic features are useful for the task, we investigated improving the coverage by adding an alternative parser.

One obvious choice would be any of the dependency parses provided by the organisers of the shared task. These parses have some advantages – broad coverage over the data and being tuned to the biomedical domain are some of the obvious ones. However, we wished to leverage off our previous feature engineering work with deriving salient indicators from RMRSs, and time constraints did nor permit us to do a full-scale investigation of using these alternative dependencies. Rather, we investigated different means of producing RMRSs using off-the-shelf components. One approach, which we mentioned as an avenue of future research in MacKinlay *et al.* (2009), is to use the broad-coverage general purpose statistical parser RASP (Briscoe *et al.* 2006). As we noted in Section 3.5.2, it is possible to produce RMRS output from RASP using a method described in Frank (2004). An implementation of this is

included with the LKB (Copestake and Flickinger 2000) which we described in Section 3.3.2, so these two off-the-shelf components can together be used an alternative source of RMRS representations of the sentences.

In our setup, we did not allow fragment analyses from RASP due to the difficulty of converting them to RMRSs outputs and doubts about their reliability, as correct analyses of well-formed scientific prose should not be fragments, with the possible exception of titles. We also, due to time constraints, did not investigate the possibility adding named entity recognition tuned to the biomedical domain with RASP as we did with the ERG, but this is definitely a possible future optimisation. Similar to our approach with the ERG, we only used the top-ranked parse.

Under these conditions, we found that RASP was able to achieve similar coverage to the ERG, obtaining a parse for 76% of the sentences in the development data. However, this set of sentences was unsurprisingly not a complete overlap with those that are parseable by the ERG. By taking the union of these sets, we increase the number of sentences for which we can produce an RMRS to 93% of the development set (as well as being able to produce RMRSs from multiple sources[2] for 58% of sentences if we wish to), making features derived from RMRSs a far more realistic prospect as a means of detecting event modification. It should be noted that occasionally, probably as a result of a bug in the conversion process or slight incompatibility in the RASP output, the RASP RMRSs contained invalid arguments for the elementary predicates, which were ignored. Additionally, on rarer occasions, with both RASP and the ERG, for reasons we have not been able to determine, invalid RMRS representations were produced which we could not use.

Combining RMRSs from different sources

Given that we have multiple potential sources of RMRSs to create feature vectors, there are several possible ways to combine them which we evaluated in our experiments. The first is a fallback method. We have more confidence in the ERG parses and their ability to produce RMRSs for a number of reasons: the ERG is a deeper grammar in contrast to the deliberate shallowness of the RASP grammar, so we would expect that where it can find a parse that its analyses would contain more useful; additionally RMRS is closer to a native format for the ERG rather than a post hoc conversion as it is for RASP. The increased confidence in the ERG is also justified empirically, as we shall see in Section 6.5. On the basis of this, it seems that a sensible strategy might be to use the ERG-derived RMRS where it is available, and where it is not, to fall back to the RMRS derived from the RASP output. In each case, we would get an RMRS that can be used for constructing feature vectors, and we would

[2]This is not to be confused with an orthogonal method for producing multiple RMRSs, by using more than just the top-ranked parse from a parsing component, which is not something we investigated in this chapter.

hope that there would be enough in common between the different RMRSs that they are at least able to produce compatible features for machine learning.

The alternatives place equal confidence in both sources of RMRSs. Each sentence will have zero, one, or two RMRSs available. In the first alternative, where we have one RMRS, we construct features from it as usual. Where there are two RMRSs, we construct features from each in the usual way, and insert all of the features created in this way into the feature vector. Where the same features are produced from each (as we would expect if the RMRSs produced were similar) the feature vector only stores the one positive value for the feature, but for non-overlapping features, both will be present in the vector and not distinguished from each other.

A variant of this method produces the same merged RMRS-derived features output if there are multiple input RMRSs, but also produces a version of each feature that is tagged with the source of the RMRS. If the ERG and RASP produced two identical RMRSs, any features that could be constructed from these would result in three positive values in the merged vector – one plain feature and one for each of the RMRS sources. The intuition here is that while there are some commonalities between the RMRS outputs, each grammar may have different strengths and weaknesses in terms of producing RMRSs, so it may be useful for the machine learning algorithm to have (indirect) knowledge of which grammar produced the particular feature.

6.3.6 Bag-of-words Features

To evaluate the performance boost we obtained in Task 3 relative to more naive methods, we also experimented with feature sets based on a bag-of-words approach with a sliding context window of tokens on either side of the token corresponding to the trigger, as determined by the tokenisation of the GENIA tagger, without crossing sentence boundaries.[3] We evaluated a range of combinations of preceding and following context window sizes from 0 to 5 (never crossing sentence boundaries). There are features for tokens that precede the trigger, follow the trigger, or lie anywhere within the context window, as well as for the trigger itself. A variant of this is to extend the lists of preceding tokens to the beginning and end of the sentence, so the features derived from the tokens are undifferentiated within the sentence except for whether they precede or follow the trigger token.

The bag-of-words context-window is robust and gives 100% coverage, so it gives us a chance at classifying the sentences which aren't parseable using either of our chosen processing pipelines. It is also possible that even on sentences we can parse with the ERG and/or RASP, the event modifications it can detect are at least partially complementary to those that are detectable with the RMRS-derived features. This means the most sensible strategy may be to combine the outputs. A fallback strategy

[3]Note that unlike the similar baseline described in MacKinlay *et al.* (2009), we do not lemmatise the tokens, or restrict it to only sentences which we were able to parse by other means

of the kind described above makes little sense here. Rather, we simply aggregate the RMRS-derived feature vectors with the most promising of the bag-of-words vectors, and allow the learning algorithm to tease out the interesting distinctions.

6.3.7 Training a Maximum Entropy Classifier

To produce training data to feed into a classifier, we parsed as many sentences as possible using the ERG and/or RASP, and used the output RMRSs to create training data using the features described above. The construction of features, however, presupposes annotations for the events and trigger words. For producing training data, we use the provided trigger annotations.

For applying the classifications over the test or development data, we have outputs from a third-party Task 1 classifier. Primarily we use the outputs of the 'UTurku' system of Björne *et al.* (2009), which we discussed in Section 2.8.2, since this was one of the best performers in the original shared task.

We also occasionally use some secondary Task 1 systems where noted in Section 6.5. One system is the CRF-based classifier of the 'NICTA' system (MacKinlay *et al.* 2009), also discussed in Section 2.8.2. In addition we use the outputs of the 'JULIELab' system (Buyko *et al.* 2009), the second-ranked system in the original shared task, which uses dependency representations of the sentences. These dependency graphs are 'trimmed', to remove irrelevant information such as modal verbs, and augmented with semantic class annotations. After this trimming stage, a feature-based maximum entropy classifier (using lexical, chunking and dependency path features) and a graph kernel classifier are used to classify events and their arguments.

For the feature engineering phase over the development data, we also have the option of using the oracle gold-standard annotations from Task 1, enabling us to focus purely on the Task 3 modification detection.

This pipeline architecture places limits on annotation performance when we are using an imperfect Task 1 classifier: the recall in Task 1 indirectly constrains the Task 3 recall, since if an event is not detected, we cannot later classify it as negated or speculative (however, if the event is not modified according the gold standard, it does not matter whether it is detected or not). Additionally, lower precision in Task 1 tends to mean lower precision in Task 3, since false positive events from Task 1 will be penalised again in Task 3 if they are classified as undergoing modification.

Here, we use a maximum entropy classification algorithm (of the kind outlined in Section 2.3.1), as it has a low level of parameterisation and is a solid performer in NLP tasks. The implementation we use is Zhang Le's MaxEnt Toolkit[4] with default parameters — specifically, the L-BFGS learning algorithm (which is very similar to the LMVM algorithm mentioned in Section 2.3.1) and with Gaussian priors (also mentioned in Section 2.3.1) disabled.

[4] http://homepages.inf.ed.ac.uk/lzhang10/maxent_toolkit.html

N1	NegOutscope2+Conj, NegConjIndex
N2	*N1*, TrigPredProps
N3	*N1*, Arg0NegOutscopeeSA
N4	*N3*, TrigPredProps, NegVOutscope
N5	*N3*, NegVOutscope
S1	SpecVObj2+WN+Conj, AnalysisSA
S2	*S1*, TrigPredProps
S3	*S1*, ModAdj, ModalOutscope
S4	*S3*, TrigOutscopes
S5	SpecVObj2+WN+Conj, ModAdj, ModalOutscope,TrigOutscopes

Table 6.2: Task 3 feature sets

Mod	**Features**	R	P	F
Negation	N2	19.6	61.8	29.8
Negation	N3	15.9	68.0	25.8
Negation	N4	19.6	67.7	30.4
Negation	N5	16.8	69.2	30.1
Speculation	S2	15.8	83.3	26.5
Speculation	S3	18.9	78.3	30.5
Speculation	S4	17.9	94.4	30.1
Speculation	S5	17.9	100.0	30.4

Table 6.3: Results over development data for Task 3 using gold-standard event/trigger. Feature sets are described in Table 6.2

6.4 Feature Engineering

In the initial stages of experimentation, we considered the large range of feature sets discussed in Section 6.3.4. It seemed likely that some combination of these features would be useful, but it was not clear what the optimal combination would be. These feature engineering experiments used only features from ERG-derived RMRSs, without any features from RASP RMRSs or the bag-of-words baseline. The combinations of feature sets we explored are shown in Table 6.2.

In Table 6.3 we present the results over the development data, using the provided gold-standard annotations of trigger words. The gold-standard figures are unrealistically high compared to what we would expect to achieve against the test data, but they are indicative at least of what we could achieve with a perfect event classifier.

Negation	**Speculation**
NegOutscope2+Conj	SpecVObj2+WN+Conj
NegConjIndex	AnalysisSA
Arg0NegOutscopeeSA	ModAdj
TrigPredProps	ModalOutscope
NegVerbOutscope	

Table 6.4: RMRS-derived feature sets used in the reported results

On the basis of these results, we selected what seemed to be the most promising feature sets for Negation and Speculation. We hold the feature sets constant for the remainder of this chapter, and investigate the effects of providing more reliable parses with higher coverage to create RMRSs from which we can derive feature values.

For Speculation, we selected the feature set denoted 'S3' in Table 6.2 and in MacKinlay *et al.* (2009), and for Negation, we use the set we denoted 'N4'. These feature sets are also explicitly listed in Table 6.4.

6.5 Initial Results

Having established a feature set to use in the various experiments, we moved on to examining the contribution of different parsers to the results. We test the impact of the two parsers — individually and in combination — on Speculation and Negation event modification over the provided development data and test data. We use the development set for feature engineering — both to select a basic RMRS-derived feature set as described in Section 6.4 and to experiment with different combinations of parsers to produce these features.

Evaluation against the test set is online-only, and can only be performed once per 24 hours, to preserve the sanctity of the test data, so we selected a handful of promising or interesting combinations of features from the experiments on the development data and applied the to the test set. Strictly, this does not conform to the best practice of applying only one parameter combination to the test set but the shared task guidelines encouraged several different submissions, while making sure that not too many were submitted and that the task organisers could determine how many test runs were submitted. We use the following sets of Task 1 annotations:

- The output of the UTurku system over the development set and test set.

- The gold-standard annotations, to evaluate our methods in isolation of Task 1 classifier noise, only over the development set (as oracle Task 1 annotations are not available for the test set)
- The outputs of the JULIELab and NICTA Task 1 classifiers, to explore the impact of Task 1 classifier performance on event modification, only over the development set.

6.5.1 Bag-of-words Baseline

We first carried out a series of experiments with different window sizes for the bag-of-words method over the development data, to determine the optimal window size for each subtask. We use the notation W_{+y}^{-x} to indicate x preceding words and y following words. Using gold-standard Task 1 data and optimising over event modification F-score, we found that the optimal window size for SPECULATION was three words to either side of the event trigger word (i.e. W_{+3}^{-3}), at an F-score of 48.3%. For NEGATION, the marginally wider window size of four words to the left and three words to the right (W_{+3}^{-4}) produced the optimal F-score of 53.3% over the development data (once again based on gold-standard Task 1 annotations). It is clear that this relatively uninformed baseline can perform surprisingly well. These window size settings are used exclusively in the bag-of-words experiments presented in the remainder of this paper for the respective subtasks.

6.5.2 Using RMRSs and Combinations

In Table 6.5 we present the results over the development data using the UTurku classifier and gold-standard Task 1 annotations. As an additional benchmark, we also include results for the Negex system (Chapman *et al.* 2001), a general-purpose tool for identifying negation using rules based on regular expressions, which was developed using clinical data. Recall that both the ERG and RASP have imperfect coverage over the data, meaning that in cases where bag-of-words features are not employed, the feature vector will consist of all negative features, and the classifier will fall back on the class priors to classify the instance in question.

Firstly, for the pure RMRS-based features, there are obvious differences between the methods of RMRS construction. The standalone ERG produces respectable performance in NEGATION and SPECULATION (although it is still outperformed by the bag-of-words baselines, particularly for NEGATION). In line with our predictions, the standalone ERG produces superior performance to the standalone RASP.

In terms of strategies for combining the features from different RMRSs, it seems that the fallback strategy (fb) is most effective: creating an RMRS from the ERG where possible, and otherwise from RASP produces a substantial performance boost over the standalone ERG strategy, which is consistent across SPECULATION and

NEGATION, and both the gold-standard and UTurku outputs. This is interesting as for only 17% of the sentences in the data was there a RASP parse and not an ERG parse. It seems that there is relatively good compatibility between the features produced from these different RMRSs, so that features learnt from RASP-derived RMRSs can be used for ERG-derived RMRS output and vice versa.

The strategy which combines every possible parse obtained from the ERG and RASP (cb) is generally less effective, with the one exception of NEGATION, where bag-of-words features are combined with the RMRS features. In fact, in the majority of cases, cb without bag-of-word features is inferior to using the ERG as a standalone parser.

When we combine the bag-of-words features with the RMRS-derived features, the results always improve over the equivalent RMRS results without bag-of-words, with recall being the primary benefactor. The cb strategy appears to benefit most from the addition of the bag-of-words features.

Comparing our results over the NEGATION subtask to Negex, it is evident that all results incorporating the ERG and/or bag-of-words features outperform this benchmark rule-based system, which is highly encouraging.

We were surprised by the effectiveness of the bag-of-words approach in comparison to our more informed techniques, particularly for NEGATION, where the simple bag-of-words baseline was superior to all other methods when combined with the UTurku Task 1 classifier. Nonetheless, the parsing techniques are clearly shown to have some utility (bearing in mind that there are still 7% of sentences which cannot be parsed under this setup thus will not be classified correctly from RMRS-derived features). However there is possibly room for improvement in the remaining 93% of sentences which we can parse — our results in Table 6.5 are still well below 93% recall.

We have not performed any analysis to verify whether the number of events per sentence differs between parseable and unparseable sentences. Longer sentences tend to be harder to parse, and may contain a larger number of events by virtue of their length, meaning that the true upper bound may be lower than 93%.

6.5.3 Interaction between Task 1 and Task 3

There is a clear interaction between Tasks 1 and 3 in our pipeline architecture, in that if there is an error in the Task 1 output for an event where there is SPECULATION or NEGATION, we have no way of correcting that mistake in our Task 3 classifier. What is less clear is the statistical nature of this interaction. To investigate this question, we plotted Task 3 performance relative to the performance of each of the three base Task 1 systems (UTurku, JULIELab and NICTA) as well as the oracle annotations, over the various combinations of features. The results for NEGATION are presented in Figure 6.2.

It is apparent from the two graphs that the correspondence is usually monotonically increasing and roughly linear, and the relative gain in Task 3 F-score is equivalent

Mod	RMRS from	Extra	Gold			UTurku		
			R	P	F	R	P	F
NEG	—	Negex	32.7	32.7	32.7	22.6	17.8	19.9
NEG	—	W_{+3}^{-4}	51.8	54.8	53.3	25.4	33.7	**29.0**
NEG	RASP	—	12.7	35.9	18.8	5.4	26.1	9.0
NEG	ERG	—	26.4	72.5	38.7	15.4	34.0	21.2
NEG	fb(ERG,RASP)	—	35.4	66.1	46.1	17.3	34.6	23.0
NEG	cb(ERG,RASP)	—	29.1	64.0	40.0	13.6	34.1	19.5
NEG	ERG	W_{+3}^{-4}	45.4	48.5	47.0	18.2	26.3	21.5
NEG	fb(ERG,RASP)	W_{+3}^{-4}	44.6	66.2	53.3	19.1	33.3	24.3
NEG	cb(ERG,RASP)	W_{+3}^{-4}	50.9	59.0	**54.6**	21.8	32.0	26.0
SPEC	—	W_{+3}^{-3}	42.9	55.4	48.3	19.0	33.3	24.2
SPEC	RASP	—	16.7	66.7	26.7	5.5	26.1	9.0
SPEC	ERG	—	20.2	68.0	31.2	10.7	56.2	18.0
SPEC	fb(ERG,RASP)	—	25.0	61.8	35.6	13.1	50.0	20.8
SPEC	cb(ERG,RASP)	—	15.5	59.1	24.5	10.7	52.9	17.8
SPEC	ERG	W_{+3}^{-3}	45.2	59.4	51.3	20.2	34.0	**25.4**
SPEC	fb(ERG,RASP)	W_{+3}^{-3}	45.2	60.3	**51.7**	16.7	31.1	21.7
SPEC	cb(ERG,RASP)	W_{+3}^{-3}	40.5	54.8	46.6	19.0	32.0	23.9

Table 6.5: Results over the development data using gold-standard Task 1 annotations and the UTurku Task 1 system ("fb" = fallback strategy, where we use the first source if possible, otherwise the second; "cb" = use undifferentiated RMRSs from each source to create feature vectors)

for every 1% gain in absolute F-score for Task 1. In the case of both SPECULATION and NEGATION, the slope of the various curves is relatively consistent at around 0.5, suggesting that it is possible to achieve a 1% increase in Task 3 F-score by boosting the Task 1 F-score by 2%. Of course, each of the curves in these graphs are based on only four data points, and there is inevitable noise in the output, but a rough linear trend is clearly demonstrated. There is only one instance of non-monoticity — over SPECULATION using fb(ERG, RASP)+W_{+3}^{-4}, where there is decrease in Task 3 score when moving from JULIELab to the higher Task 1 score from UTurku. In addition, the slope of increase is considerable lower for SPECULATION when bag-of-words features are omitted, but this does not detract from conclusions about a linear correlation; it is unsurprising that the less effective features do not show rises which are as dramatic.

Mod	**RMRS from**	**Extra**	R	P	F
NEG	—	W_{+3}^{-4}	19.55	30.94	23.96
NEG	—	Negex	17.83	12.73	14.85
NEG	ERG	—	11.82	35.62	17.75
NEG	fb(ERG, RASP)	—	13.64	34.09	19.48
NEG	cb(ERG, RASP)	—	12.73	33.33	18.42
NEG	ERG	W_{+3}^{-4}	19.55	32.33	24.36
NEG	fb(ERG, RASP)	W_{+3}^{-4}	19.55	32.58	24.43
NEG	cb(ERG, RASP)	W_{+3}^{-4}	20.91	41.07	**27.71** †
SPEC	—	W_{+3}^{-3}	6.97	23.73	10.77
SPEC	ERG	—	8.96	41.86	14.75
SPEC	fb(ERG, RASP)	—	12.44	52.08	**20.08**
SPEC	cb(ERG, RASP)	—	6.47	41.94	11.21
SPEC	ERG	W_{+3}^{-3}	11.44	26.14	15.92
SPEC	fb(ERG, RASP)	W_{+3}^{-3}	9.95	28.17	14.71 †
SPEC	cb(ERG, RASP)	W_{+3}^{-3}	7.46	24.19	11.41

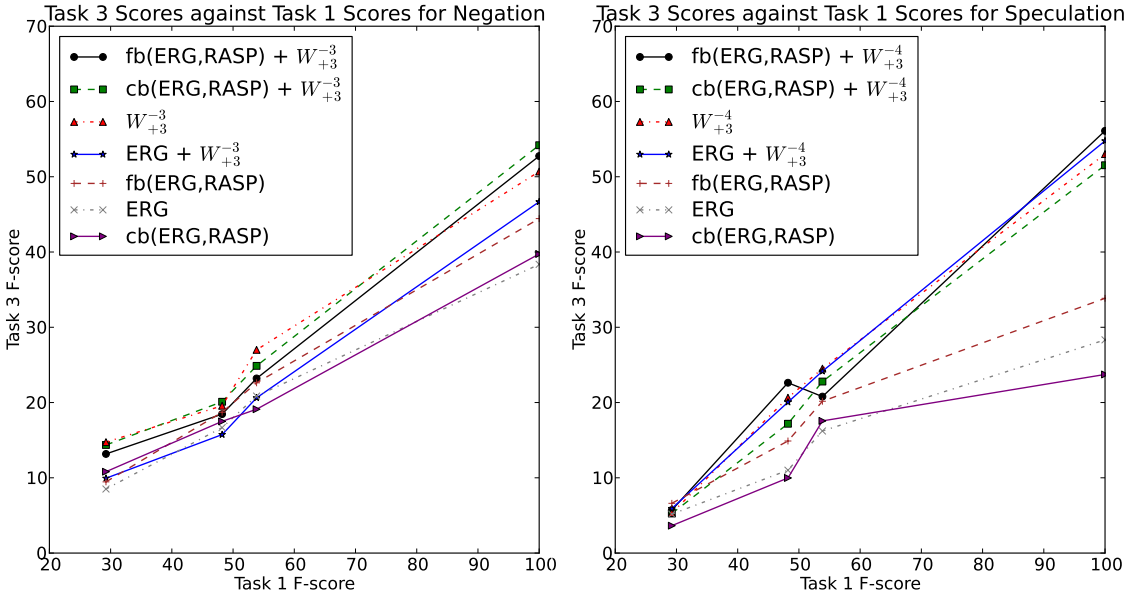
Table 6.6: Results over the test data using the UTurku Task 1 system ("fb" = fallback strategy, where we use the first source if possible, otherwise the second; "cb" = use undifferentiated RMRSs from each source to create feature vectors). † denotes the feature set which performed best over the development set using gold Task 1 annotations

6.5.4 Results over the Test Data

In the testing phase, we repurposed all of the development data as extra training data, and retrained using some of the promising combinations of RMRS sources and bag-of-words feature vectors. These results are presented in Table 6.6. Note that we are not able to evaluate over gold-standard Task 1 data, as it has not been released for the test data.

The results here are not always what we would expect on the basis of the development results. The bag-of-words baseline continues to be an impressive performer for NEGATION, achieving an F-score of 24.0% with the UTurku data compared with 29.0% over the development data. However the combination of the aggregated RMRS approaches and the bag-of-words features outperformed bag-of-words.

The SPECULATION results show noticeably different behaviour from the development data. The primary difference seems to be that the bag-of-words baseline (at least for the context window that we selected) is of little use in comparison to the RMRS features. Encouragingly, the best result was obtained with a pure parser-based



(a) Negation (b) Speculation

Figure 6.2: Task 3 Scores against Task 1 Scores

approach (fb(ERG,RASP)), and bag-of-words on its own was the poorest performer, with an F-score around half that of the parser-based method. This effect is even visible when combining the bag-of-words with the RMRS output, which resulted in a substantial decrease in F-score. Examining further, we can see that the bag-of-words recall is particularly low over SPECULATION, so it seems that the local contextual cues for SPECULATION which were learned from the training and development data are simply not present in the accessible events in the test data, while the longer distance syntactic dependencies are still clearly useful.

In terms of overall performance in comparison to the original submissions to the shared task described in Kim *et al.* (2009), these results are respectable. If we had been required to choose only one run for each of SPECULATION and NEGATION, the features would have been selected on the basis of the development set figures with gold Task 1 annotations (another option would be to use the best automatically created Task 1 annotations) — these figures are marked with '†' in Table 6.6. For SPECULATION, we would have submitted the fb(ERG,RASP), W^{-3}_{+3} system to give an F-score of 14.71% giving results higher than the second-placed team, but well behind the score of ConcordU of 25.27% (the best performer over the test set would have

been closer to the ConcordU performance, but this is not a fair comparison to make as it takes advantage of knowing scores over the test data). In the NEGATION subtask, using this technique would have selected the same parameters which gave the best test set performance, giving an F-score of 27.71% — higher than the top-ranked ConcordU score of 23.13%. Of course, in both cases these results rely on high-performing Task 1 systems from third parties which is important for Task 3 results as we discussed in Section 6.5.3.

6.6 Revisiting Parsing

The work up to this point in the chapter has largely reflected the relevant work from MacKinlay *et al.* (2011b), using the same grammar version and preprocessing. However, since the original research on which the paper was based took place, there have been several developments which have the potential to improve parsing using the ERG. Firstly, there have been a large number of grammar updates, as the ERG has been under active development. Secondly, in Chapter 5, we explored ways to create training data tuned for the biomedical domain, and showed that this could improve parsing accuracy. It seems logical to investigate whether the accuracy boost we get by training from this domain-tuned data translates into improved accuracy on a downstream task such as the detection of modification of biomedical events which we are investigating in this chapter.

6.6.1 Updating the Grammar Version

The first change we made was to update the grammar version to the version used in Chapter 5, which is slightly updated compared to the version tagged as '1010'. This is useful for two reasons. Firstly, the coverage is increased with relatively little effort from our perspective (although not, of course, for the grammar engineer). Recall from Section 6.3.1 that the coverage using the ERG for the first round of experiments was 76%. Using this newer version of the grammar, the coverage is increased to 80%. It should be noted that none of changes to the grammar in that time were specifically targeted at biomedical text. Instead, they were targeted at other domains, such as Wikipedia, for the WeScience corpus we mentioned in Section 3.4.1.

The second advantage is that this newer grammar version matches the version used for the biomedical treebank which we introduced in Section 5.3.1. This makes using this treebank for creating or evaluating parse selection models more straightforward. However, as we discuss in the next section, this is complicated somewhat by preprocessing.

6.6.2 Updating Preprocessing

We have noted more than once that preprocessing, particularly POS-tagging and handling of named entities, is particularly important for domains such as biomedical articles which use many unseen constructions, including punctuation. There are two primary preprocessing configurations we have used in this thesis. Up to this point in the chapter, the configuration was that of the original research — specifically, using the GENIA tagger for POS-tagging and named entity recognition, and treating the named entities as atomic entities. In the following discussion we denote this P_1. There is also the more complex setup we used throughout Chapter 5, which we described in detail in Section 5.3.2. This uses the GENIA tagger but defers to TnT for tagging noun-like tokens, and inputs named entities as atomic entities as well as individual tokens using a lattice input. We denote this P_2.

This is the configuration used in creating the small biomedical treebank, and also for the parse forests we used for treeblazing in Chapter 5. Using P_2 would confer some similar advantages to using the newer grammar version. Firstly, if we are already using the updated ERG, we get an increase in coverage from the figure of 80% noted in the previous section to 85%. Additionally, there are similar advantages to having outputs which are directly comparable with our pre-existing treebank. The evaluation of parse accuracy against the treebank is more meaningful, and the parse selection features learnt from the (blazed or automatically-created) treebanks are more directly compatible for applying to new models.

In initial experiments to compare P_1 and P_2, we used the new ERG version, and applied a parse selection model trained on the combined WeScience and LOGON corpora, varying only the preprocessing configuration. The feature set we used was one of the strongest performers: fb(ERG,RASP) + W_{+3}^{-4} for Negation and fb(ERG,RASP) + W_{+3}^{-3} for Speculation. Surprisingly, we found that over the whole pipeline, the F-scores for modification detection were somewhat lower using P_2 than the less sophisticated and lower coverage P_1. In fact, the configurations using P_2 sometimes obtained a substantially lower F-score than the results reported in Table 6.5, which used the older grammar with 9% lower coverage. The F-score for Negation dropped from 53.3% to 48.0%, while for Speculation there was a modest increase from 51.7% to 52.4%. More importantly, the results for P_2 were somewhat lower than using P_1 with the newer grammar, which obtained F-scores of 56.6% for Negation and 55.5% for Speculation. These results are shown in more detail in Table 6.7

It is not entirely clear why this should be the case. Naively, we would expect the higher coverage, and the ability to produce arguably better quality parse trees[5] should correspond to higher accuracy on such a downstream task. One possible

[5]Recall from Section 5.3.2 that P_2 was modified using lessons learnt of an initial iteration of treebanking to increase the chance of a better quality tree being available, since sometimes the treatment of NEs as atomic entities removed a better-quality parse

contributing factor is that the higher numbers of trees licensed by the more permissive P_2 preprocessing meant that even with a more informed parse selection model, there was more chance of having a low-quality tree with many incorrect dependencies as the highest ranked. Another is that since we did most of the development using P_1 (albeit with the older version of the grammar), the features we used were implicitly tuned to match the RMRS output resulting from this configuration.

Regardless, the purpose of this section is to investigate whether a better parse selection model can improve downstream performance, and secondarily, whether there is a correlation between parse selection accuracy and accuracy for detection of event modification. P_1 is a better fit for the first goal, while both are equally valid for the second, so we conduct most of our experiments using P_1. The primary caveat relates to the point noted above, about the ideal evaluation using the same preprocessing configuration as the treebank. If this does not match, the evaluation is still meaningful, but we would expect a performance drop compared to matching configurations. However, this should affect different parse selection models approximately equally, so the parse selection metrics should still be valid for relative comparison of the parse selection models, even if the absolute scores look somewhat lower than we saw in Chapters 4 and 5.

6.6.3 Applying Better Models for Biomedical Parsing

We have various combinations of corpora which we discussed in the earlier chapters which are adaptable for use as training corpora. In Chapter 4, we made extensive use of the WeScience and LOGON corpora. The experiments in the earlier part of this chapter used a previous iteration of the LOGON corpus, and this is still available to us. The more recent WeScience corpus is also available, and intuitively we would expect this corpus to be a closer match to the domain of biomedical abstracts, although in the following section we evaluate this empirically.

There are also several sources of training data from Chapter 5 with different expected levels of impact on parse selection accuracy. We should expect the biggest accuracy boost from the ERGGTB biomedical development treebank of Section 5.3.1. However since this is a relatively small dataset, we will need to use it conjunction with other training data, and upweight the data using the Duplic method we explored in Chapter 4. We upweight the biomedical treebank by a factor of 8 when training parse selection models, since we found that to be a strong performer. We also use this treebank as we did in Chapter 5, to evaluate parse selection accuracy. When we are using it as a training corpus as well, we use 8-fold cross-validation to evaluate the parse selection accuracy. No changes are necessary when applying the model to the BN09ST experiments themselves, since the corpus by design does not overlap with the shared task data.

Naturally, we can also use the blazing and self-training techniques from Chapter 5, where we use automatically created training data of some kind (in combination with

a manually annotated corpus such as WeScience). While we are using a different preprocessing method, it is straightforward to create new parse forests for self-training and blazing which use the same P_1 configuration, and train models in the usual way from this newer data. This may create models which are better adapted for parsing using the matching preprocessing method which we use for parsing the shared task data itself. Note, however, that since we are evaluating against a treebank which uses P_2, it may not be meaningful to compare the parse selection scores from a blazed/self-trained model using a non-matching P_1 configuration to create the training forest with those from models created using a P_2-derived training forest.

Before building models using these techniques, it is necessary to filter out those sentences which overlap with the shared task data, since roughly half of the sentences are also used in the annotated abstracts from the shared task.[6] We were able to partially offset the loss in training data by adding sentences which had originally been held-out from experimentation in Chapter 5 although overall there were still fewer sentences available – 4929 parseable by the ERG (and thus potentially usable as training data) against 10747 in Chapter 5. The loss in data was slightly exacerbated by the lower-coverage preprocessor configuration we were using.

6.6.4 Results with Newer Models

We created parse selection models from various sources of training data as discussed in the previous section. The training corpora were combinations of WeScience, LOGON, ERGGTB, blazed training data and self-trained training data. The parse forests for blazing and self-training (i.e. before filtering to create training data) use either the P_1 or P_2 preprocessing configuration. For each parse selection model, we calculate the parse selection accuracy (using cross-validation where required on ERGGTB) and EDM F-score, and compare this to the scores we obtain by parsing the shared task sentences using that parse selection model.

We present the results over the BN09ST development data in Table 6.7, using oracle Task 1 annotations as well as the UTurku annotations. We would naively expect an increase in parsing accuracy and F-score to correspond to an increase in Task 3 performance, however these experiments do not show this to be a strong effect. The blazed model Blaze:P_2(WS) is not even augmented with any explicitly manually-annotated data unlike all other models in the table, which include either WeScience or LOGON— it uses only the data from a standard treeblazing configuration. In line with the findings of Chapter 5, this achieves fairly low EDM and Acc_1 scores. However, this gives the best or near-best parse selection results using the UTurku Task 1 annotations, and mid-range scores with gold Task 1 annotations. In contrast, the blazing and self-training configurations which append manually-annotated data

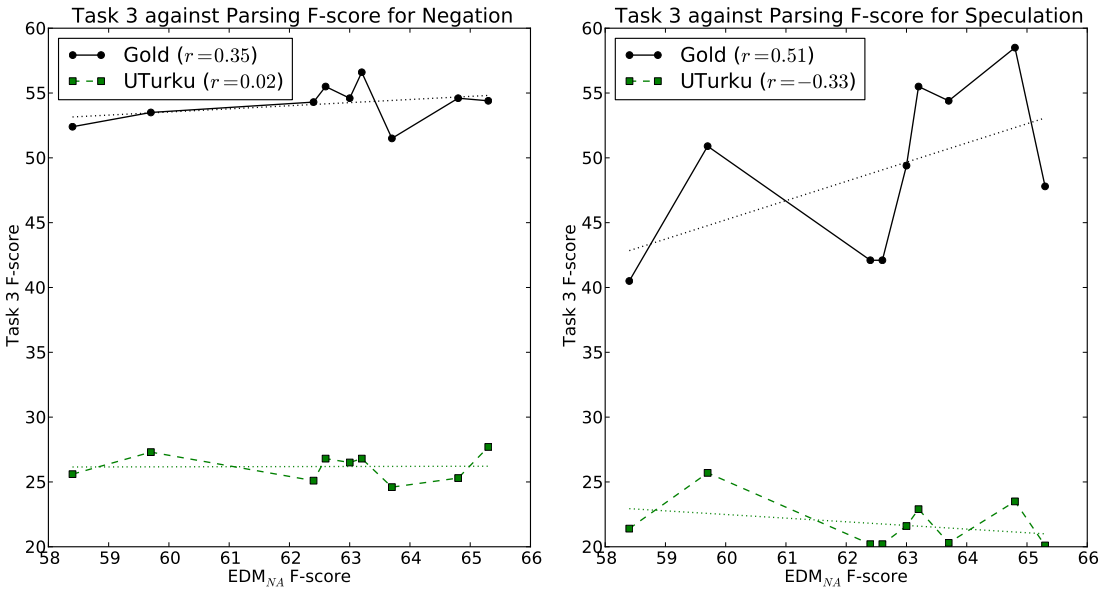
[6] With self-training, we could easily add extra data from additional unannotated research abstracts, since there is no requirement that the data have any existing annotations at all but we avoided this for comparability and simplicity of experimental design.

Model	Acc_1/EDM	Negation				Speculation			
		Gold		UTurku		Gold		UTurku	
		R	P	F	R	P	F	R	P	F	R	P	F
LOG (Legacy Conf)	NA/NA	44.6	66.2	53.3	19.1	33.3	24.3	45.2	60.3	51.7	16.7	31.1	21.7
LOG	3.1/58.4	51.4	53.4	52.4	24.3	27.1	25.6	36.8	44.9	40.5	16.8	29.4	21.4
Blaze:P_2(WS)	3.1/59.7	50.5	56.8	53.5	25.2	29.7	27.3	44.2	60.0	50.9	20.0	36.0	**25.7**
WS	5.7/62.4	50.5	58.7	54.3	22.4	28.6	25.1	37.9	47.4	42.1	15.8	28.0	20.2
S-T:P_1(WS)+WS	6.4/62.6	52.3	59.0	55.5	24.3	29.9	26.8	37.9	47.4	42.1	15.8	28.0	20.2
Blaze:P_1(WS)+WS	6.0/63.0	52.3	57.1	54.6	24.3	29.2	26.5	46.3	53.0	49.4	16.8	30.0	21.6
WS +LG	6.3/63.2	52.3	61.5	**56.6**	24.3	29.9	26.8	50.5	61.5	55.5	17.9	32.0	22.9
Blaze:P_2(WS)+WS	6.0/63.7	49.5	53.5	51.5	22.4	27.3	24.6	48.4	62.2	54.4	15.8	28.6	20.3
EG ×8 + WS	6.9/64.8	49.5	60.9	54.6	22.4	28.9	25.3	50.5	69.6	**58.5**	17.9	34.0	23.5
EG ×10 +WS +LG	7.8/65.3	52.3	56.6	54.4	25.2	30.7	**27.7**	45.3	50.6	47.8	15.8	27.4	20.1

Table 6.7: Results using fb(ERG,RASP)+W^{-4}_{+3} for Negation and fb(ERG,RASP)+W^{-3}_{+3} for Speculation with P_1 preprocessing and various parse selection models. Parsing accuracies are over ERGGTB (abbreviated EG), the ERG treebank of GTB sentences from Section 5.3.1. WS denotes WeScience. 'EG ×8 +WS' indicates the development corpus combined with WeScience using Duplic with a weighting of (8, 1), for which parsing figures are calculated using 8-fold cross-validation. 'Legacy Conf' refers to results from the parsing configuration in Section 6.5.2 (reproduced from Table 6.5), which uses an older grammar than ERGGTB, so we cannot calculate meaningful parsing accuracy.

from WeSc produce better parsing performance (relative to blazing alone or WeSc alone), but can have either a small positive or small negative impact on Task 3 performance. The WeSc +LOG model has surprisingly high Task 3 accuracy in some cases (with the best results for gold over Negation), despite the fact that it uses no biomedical training data at all.

Overall, the evidence for the value of the domain-tuned training data (either manually annotated or blazed) on this downstream task is equivocal, despite the reliable increases in parsing accuracy and F-score. Two of the best F-scores come from models which use hand-annotated data from ERGGTB, but there is no guarantee of a reliable improvement. In particular, EG ×10+WS +LG performs quite poorly over Speculation, with mid-ranging performance using gold Task 1 annotations and the lowest performance use UTurku annotations. Most models are more reliable over negation, in that there is a smaller range of F-scores, but as we saw from the results in Section 6.5.2, the bag-of-words based features are probably providing much of the accuracy here, so the quality of parses may have relatively little impact.



(a) Negation

(b) Speculation

Figure 6.3: Task 3 Scores against EDM_{NA} scores for parsing. Dotted lines show best fit, with the correlation coefficient r shown in the legend

The lack of a trend in Task 3 performance compared to parsing EDM F-score is clearer from the graphs in Figure 6.3, where we have plotted them against each other. We have additionally added regression plots (with the correlation coefficient r shown in the legend) to the graphs to make the trends clearer. The strongest trend is for

SPECULATION using gold Task 1 data, but even this is fairly weak at $r = 0.51$, while the other configurations have weaker and sometimes negative correlations. None of the slopes are significantly different from zero. Comparing against exact match tells a similar story, with weakly positive or negative correlations, but none being statistically significant. If there is an effect from the improved parsing models, it is insignificant compared to random fluctuations as a result of an imperfect machine learning process and a fairly small dataset.

This task may not have been the ideal setting for a task-based evaluation of parsing. For one thing, only a very small percentage of EDM tuples in a given corpus participate in features which are linked to actual instances of modification — firstly because there are few instances of modification compared to the number of sentences (202 modification instances in the development corpus of 1464 sentences), and secondly because even in sentences containing modification, many MRS elements will not be used in the feature extraction. Even with the difference between our best and worst parse selection models of 6.9% in EDM F-score, many of the tuples that were corrected by the better model would have no chance of being used in the Task 3 classification model. Another factor is that there is probably a large amount of redundancy in the features (particularly alongside the bag-of-words features), so for a more accurate parse to improve Task 3 accuracy, it would need to not only get a relevant MRS element correct when it was wrong with the poorer parse selection model, but the extra information it provides would need to be not duplicated implicitly by some other feature.

For more sensitivity to parse selection models, we would probably want fewer fallback features not based on parsing, or a downstream application which makes usage of a higher percentage of the relationships in the MRSs. An explicitly rule-based approach, rather than an approach based on machine learning, would also be likely to show stronger effects without the robustness introduced by the machine learning algorithm. Miyao *et al.* (2009), for the alternative information extraction task of protein-protein interaction (PPI), found that a 1% increase in parse selection accuracy (obtained by using a better parser for the domain) corresponded to a 0.25% increase in accuracy in the downstream PPI task, so we might expect a similar conclusion to hold if we applied RMRS-based features to such a task. The PPI task has more in common with Task 1 here than Task 3, so applying RMRS-based features to Task 1 could show similar improvements with a better parse-selection model. Another hypothetical task which might show more benefits from a better parse selection model is a general purpose information extraction engine, which extracts a large number of semantic dependencies from the text to construct a database of information. With a larger number of tuples involved in the feature creation, the effects of improving parse selection accuracy would be correspondingly more noticeable, and the effects of noise from a relatively small number of training/test instances would have less of an effect.

6.7 Summary

In this chapter, we presented a method for detecting modification (in the form of SPECULATION or NEGATION) of biomedical events mentioned in research abstracts, as defined in Task 3 of the BioNLP 2009 Shared Task. The system used a pipeline architecture, where we first used some existing Task 1 system to detect event trigger words, and fed this as input into our Task 3 system for detecting event modification.

The Task 3 system used the semantic output of the ERG and/or RASP, and created a selection of linguistically-rich feature vectors from this output, which was fed to a maximum-entropy machine-learning algorithm. The features for detecting modification were a combination of carefully-tuned features to match instances of modification seen in the training data and general purpose features to take advantage of the robustness of machine learning algorithms. A bag-of-words approach based on a sliding context window of 3–4 tokens on either side was a strong performer as a baseline, and these feature vectors were also used to augment the parser-derived features in the most successful configurations. This bag-of-words baseline was surprisingly effective, producing the best performance for NEGATION over the development data, although the hybrid approach with parser and bag-of-words features was still an improvement for SPECULATION.

Over the test set, the bag-of-words baseline was less effective. For NEGATION, the parser-based approach gave a small improvement, while for SPECULATION, there was an even more dramatic drop in the usefulness of the bag-of-words features, to the extent that they had a detrimental effect when added to the parser-based features, which seemed to be more robust. The best-performing feature set was a system with only parser-derived features, although on the basis of the development set results, we would have chosen a hybrid system, which still improved greatly on the bag-of-words F-score.

We also found from testing over a range of Task 1 systems that, as we would expect in a pipeline architecture, the Task 3 performance is greatly dependent on Task 1 performance. It is difficult to detect modification of events when the events themselves have not been identified correctly. One of the best ways to improve Task 3 performance is to improve Task 1 performance, and the size of this effect eclipses most of our careful feature engineering for Task 3.

In another round of experimentation we investigated the effect of the parse selection model on Task 3 performance. A higher parse-selection F-score corresponds to more accurate RMRSs, and we would naively expect this to lead to extracting more accurate features from these RMRSs, and a correspondingly higher score on Task 3 which uses the outputs of these features. However, we did not find a significant correlation between the parsing F-score and Task 3, possibly because our Task 3 system was not an ideal system for seeing the effects of parse selection accuracy on downstream applications.

In this chapter we have examined the application of deep parsing to a domain-customised information extract task. In the following chapter, we move on to a more general purpose question, investigating how to abstract away from spurious differences that are observed in the semantic output of a deep parser.

Chapter 7

Learning DMRS correspondences

7.1 Introduction

We have already seen in this thesis that precision grammars such as the ERG are able to produce detailed, precise semantic representations from input text in the form of MRS or a related formalism. However, such representations can lead to insufficient robustness when the semantic representations are compared. Such comparisons are often useful in particular target applications, such as question answering, machine translation, paraphrasing and information extraction, and robustness is often a desirable characteristic in such applications. Many of these systems rely on shallower features based on a bag-of-words–style approach or similar to obtain the necessary robustness. Indeed, in Chapter 6, we saw that shallow word-based features were required in addition to parser-based features for optimal accuracy in an information extraction task. This lack of robustness is likely to be a result of the high level of detail encoded in the semantic representations, as they are able to indicate very fine-grained differences.

The information encoded by these detailed representations is linguistically motivated. However, the encoding of particular fine-grained distinctions may obscure useful commonality which would be more easily picked up by a shallower approach. When we are comparing the semantics of two different sentences, there is thus a chance of spurious false negatives; these fine-grained representations may make the sentences appear further apart semantically, when a human may judge the sentences, or subparts of the sentences, to be very closely related. For example, the MRSs (and, correspondingly, the RMRSs and DMRSs) for *aspirin synthesis* and *synthesis of aspirin* are quite different, despite obvious semantic similarity to a human reader. Indeed, in this particular case, a bag-of-words similarity metric would also assign a high similarity score, so using the complete outputs of the precision grammar would put us at a disadvantage compared to the less-sophisticated approach.

If we wish to retain the precise ERG analyses in cases such as this, but also match superficially different sentences or sentence fragments, we require some form

of robust inference to equate these distinct structures. In this chapter we describe some preliminary work (less mature than the other work described in this thesis) which attempts to add such robustness for the semantic representations of particular kinds of sentential elements. We investigate acquiring "robustness rules" from a parsed corpus, using the notion of *anchor text* (Lin and Pantel 2001) to discover sets of semantic relationships which have very similar meanings. This relies on the assumption that if we frequently observe distinct semantic paths linking very similar nodes, there is evidence for relatedness of the paths.

We focus on the DMRS formalism described in Section 3.5.3. This graph-based representation is well-suited to the discovery of such rules using these techniques, since it has a clearly-defined notion of paths between nodes. Our search for inference rules can be recast as a search for DMRS paths which can connect pairs of nodes, and which are close equivalents in terms of how they would be interpreted by a human. Since DMRS is freely convertible to MRS (and vice versa), all of the rules are in principle applicable to MRS, but DMRS provides a more convenient framework for performing and describing such inference. This is primarily because of the aforementioned graph structure of the DMRS — it is more natural to think about subgraphs (groups of nodes and labelled links) than it is to think about groups of EPs with shared arguments or outscoping relationships, as would be required in MRS. A secondary motivation for using DMRS was a desire to explore the capabilities of what was, at the time of experimentation, a relatively novel formalism.

There has been previous work on manually constructing relationships between MRSs for several reasons, such as semantic transfer in machine translation systems (Flickinger *et al.* 2005b), and for resolving temporal expressions (Schlangen 2003: Chapter 9). Our work here differs in that we explicitly use the graph-based DMRS representation, and that we use a corpus-based approach to derive the inference rules. In the absence of a task-based evaluation, in this work, we evaluate the inferred rules by applying them to unseen DMRSs and using these for tactical generation, to provide evidence that the resulting DMRSs are well-formed, and thus that the rules are valid.

7.2 Related Work

The approach we present in this chapter is somewhat unique, partially because much of the work here is concerned with handling particular aspects of a relatively new semantic formalism. Nonetheless, some related work exists, some of which was the inspiration for the work presented here. One important related task is *paraphrase generation*. In this task the goal is, given some input string, to create variants of the string with a different surface form but which are semantically similar. This has some parallels with the work we report here, as will become clear below. However,

we emphasise that this work is not itself paraphrase generation, although that is one possible use to which it could be put.

The most relevant work for this chapter is that of Lin and Pantel (2001), who present an unsupervised approach for learning inference rules in their system 'DIRT' (Discovering Inference Rules from Text). The goal is to infer rules such as ***X*** *is the author of* ***Y*** $\approx$ ***X*** *wrote* ***Y***. They describe their approach as slightly broader than paraphrasing, since some of the rules they intend to learn will not be exact paraphrases but can nonetheless be loosely inferred from the text — hence the term 'inference rules'. The work is based on dependency parses of the sentence, and the inference rules learnt take the form of mappings between dependency paths, which are subgraphs of the complete dependency graphs produced for whole sentences. The mappings are created on the basis of what they call the Extended Distributional Hypothesis — "If two paths tend to occur in similar contexts, the meanings of the paths tend to be similar" (Lin and Pantel 2001:326). From the various corpora, they gather statistics of dependency paths, each consisting of words and dependency labels, and relating two word endpoints, which are denoted *slots*. They record dependency paths together with counts of the words which have occurred in each endpoint slot in the corpus, and then use Mutual Information statistics to compute the most similar paths. The intended use of this is in a Question Answering (QA) system, so they perform a manual evaluation over questions from a QA shared task, extracting dependency paths from the questions, and then the corresponding most similar 40 paths to those dependency paths. Each set of top 40 paths was evaluated as correct if it could plausibly appear in a answer to the original question from the shared task, with accuracies for most questions falling between 35% and 57.5%.

Many other approaches have been presented to address broadly-related problems. Pang *et al.* (2003), investigating what they call 'paraphrase induction', present an approach based on inducing finite state automata (FSA) from sets of parallel sentences. The data comes from alternative English translations of Chinese sentences from a Machine Translation (MT) shared task, and from the parses of the equivalent sentences, they create a structure similar to a parse forest by merging nodes, which is then converted into an FSA. They propose the output of the system could be used for MT evaluation as well as in QA systems. Barzilay and Lee (2003) examine a similar problem, but are primarily concerned with domain-specific paraphrases of whole sentences — for their target application, the sentential context is required. Rather than using parallel corpora, they used *comparable* corpora — news articles on similar topics, and they then align the sentences from these corpora and learn paraphrasing templates by comparing the aligned sentences. More recently Wubben *et al.* (2011) noted that paraphrasing can be thought of as a monolingual MT task (where the source and target language are the same), and compare syntax-based and phrase-based approaches to the problem.

Moving on from paraphrasing, Recognising Textual Entailment (RTE: Dagan and Glickman (2004)) is a separate NLP task, where the goal is to determine, given a

hypothesis H and a text T, whether H is entailed by T. This is not strict logical entailment, but rather entailment as it would be interpreted by a human. As such, it requires background knowledge as well as a somewhat looser interpretation of entailment than would be expected in a strict logical inference task. It is quite distinct from paraphrasing, but certain techniques appear to be applicable to both tasks. Much of the research in RTE has be driven by shared tasks (Dagan *et al.* 2006; Giampiccolo *et al.* 2007 *inter alia*) providing standardised datasets. The kinds of inference rules we study in this chapter have a possible application in this field, and indeed, some work exists which uses similar approaches. Dinu and Wang (2009) apply the rules from DIRT system described earlier in this section to the textual entailment task. The precision is a significant improvement over a naive baseline, although the coverage of the rules was too low to be useful in this fairly preliminary work.

7.3 DMRS Inference Rules

7.3.1 DMRS Overview

We have already given a high-level outline of MRS and DMRS in Section 3.5. In this section we briefly describe some more characteristics of DMRSs which are relevant to this chapter with the aid of some examples. In Figure 7.1, we show sample DMRSs for some noun phrases which happen to be semantically similar, although this fact will not become relevant until the Section 7.3.2 (where we repurpose them to motivate the need for DMRS inference). In the DMRSs displayed, note that in each case, there are *lexical predicates*, `_synthesis_n_of` and `_aspirin_n_1`, which are preceded by an underscore. These correspond to open-class words in the lexicon which can be matched to an overt substring of the sentence, and either correspond to entries in the manually-created lexicon or automatically constructed predicates for out-of-vocabulary items.

We also have *construction predicates* without a leading underscore, which generally come from a small set of closed-class lexical items, or from syntactic constructions. The construction predicates are used only when required to make the semantics coming from the lexical entries well-formed. For example, `compound` is used for noun compounds, since they do not in general have intersective semantics as we would see with adjectives modifying nouns. There are 100 possible construction predicates in the '0907' version of the ERG we use in this chapter. These construction predicates are often modelled structurally on lexical predicates — for example, `compound` has a deliberately similar structure to prepositions such as `for_p`. In particular, the head noun (`_synthesis_n_of` in these cases) is `ARG1` while the dependent argument is `ARG2`.

Construction predicates are also used heavily for quantifiers. All predicates indicating nominals in a DMRS produced by the ERG have a corresponding quantifier.

This may be a true quantifier, such as `_every_q`, a determiner such as `_the_q` or a construction predicate such as `udef_q`. The restrictors of these quantifiers are related to the predicate nodes by links such as `RSTR/H` in Figure 7.1 (these correspond to qeq conditions in the MRS). The `udef_q` quantifier is essentially a default quantifier, used in many cases where there is no overt quantifier in the sentence, such as bare plurals and bare mass nouns. It is underspecified, as the interpretation of the quantifier depends on context, which aligns with the goal of the ERG to not introduce spurious ambiguity in the semantics which does not correspond to syntactic ambiguity (Copestake *et al.* 2005). The handling of quantifiers is fairly important for the techniques we present in this chapter, and we return to this issue in Section 7.5.1.

7.3.2 Motivation

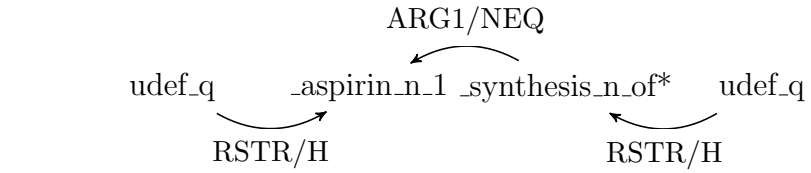
As a concrete illustration of the possible utility of DMRS inference, we return to the DMRSs shown in Figure 7.1. These all refer to semantically similar variants on *synthesis of aspirin*. It should be clear that while all the phrases appear semantically similar to a human reader, an exact comparison of the DMRSs would not produce a match.

We cannot simply ignore the label of one of the predicates such as `_for_p` to produce a match, as there are differences in other predicates as well as the number of predicates in total. In Figure 7.1(a), there are fewer predicates, as *synthesis* is treated as a relational noun taking a single argument. However, in Figure 7.1(b), it cannot be known to be relational since the possessive construction is semantically ambiguous (as we see in *Hoffman's synthesis*), so the underspecified `poss` relation, which allows either possibility, is used instead. Another example of underspecification is shown by the `compound` relation, which does not attempt to define the actual nature of the relationship.[1]

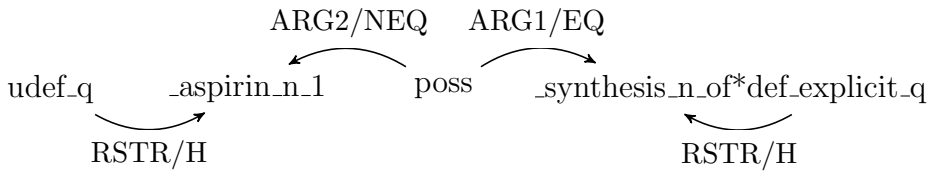
The differences between the DMRSs are semantically motivated, but nonetheless complicate the task of matching the graphs. In common between the various DMRSs is the fact that the paths joining `_aspirin_n_1` and `_synthesis_n_of` are composed of fairly "semantically light" predicates: construction predicates, and prepositions. Paths composed of predicates such as these are thus good candidates for being semantically equivalent. In addition, we consider *light verbs* such as *do* in *do the shopping* and *take* in *take a shower* (Stevenson *et al.* 2004 *inter alia*). Collectively, we denote these predicates *light predicates*, although we do not precisely specify the definition of these here.

We should note at this point that a deep grammar such as the ERG does more work in extracting semantic relationships than is found in typical schemes of grammatical

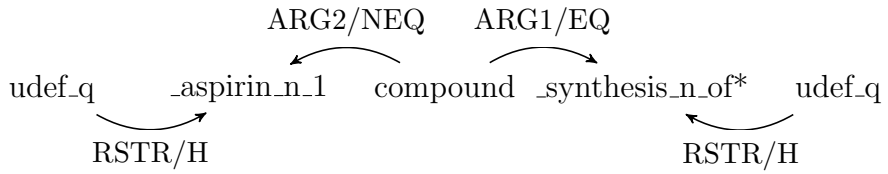
[1] Determining the nature of this relationship is an object of research in its own right — see, e.g. Lauer (1996) — and is thus beyond the scope of what the grammar can be expected to do.



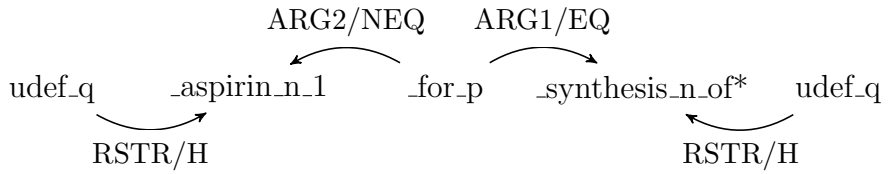
(a) *synthesis of aspirin*



(b) *aspirin's synthesis*



(c) *aspirin synthesis*



(d) *synthesis for aspirin*

Figure 7.1: Some superficially different DMRSs for semantically-related phrases; '*' denotes the head of the phrase.

relations. For instance, the preferred parses for (7.1)–(7.4) all contain the predicate `_synthesize_v_1` linked to `_aspirin_n_1` by a link labelled `ARG2`.

(7.1) *Hoffman synthesized aspirin.*

(7.2) *Aspirin was synthesized by Hoffman.*

(7.3) *Synthesizing aspirin is easy.*

(7.4) *Synthesized aspirin is effective.*

Nonetheless, as the earlier examples show, there is still considerable variation in certain DMRSs we would like to think of as semantically related in many circumstances.

Thus for some applications, we would argue that DMRS does not abstract away from the surface syntax sufficiently (even though it provides a little more abstraction than some other semantic representations). However, we cannot expect every syntactic formalism to have exactly the right level of abstraction for every task — if DMRS was made more abstract to ensure there was more in common between the above examples, then we would lose the ability to differentiate them in ways which could be important for other applications (not to mention potentially complicating the creation of these meaning representations as part of the parsing process).

In this chapter, we attempt to construct inference rules over the DMRS which are somewhat tuneable, enabling the addition of an appropriate level of abstraction for the particular task, avoiding the need to make major changes to the internals of the semantic representation just because it needs to be applied to a new task. We wish to score DMRSs such as those in Figure 7.1, as well as other similarly-related pairs of DMRSs, as matches or partial matches to each other. This can be thought of as analogous to giving a higher matching score to synonyms when comparing pairs of words, but over the space of semantic paths instead of words. Generally we look for correspondences between paths of zero or more light predicates (we discuss these conditions further in Section 7.3.3) but this requires some way to relate the distinct paths. Constructing such rules by hand is infeasible, because there is a large number of potential semantically similar paths of light predicates when longer paths are considered, and the matches are approximate and context-dependent. Here, we aim to discover the inference rules automatically, based on corpora of parsed text, to allow us to recognise closely equivalent paths, while not producing unwanted matches. It should also be possible to apply the inference rules to given DMRSs to produce a new, well-formed DMRS (and correspondingly a well-formed MRS).

7.3.3 Structure of DMRS Inference Rules

We are interested in principle in the general case of inference rules (or mappings) between two DMRS graphs, G_1 and G_2. These mappings should approximately preserve meaning and be applicable over a range of contexts. However, this fairly general

formulation, which may in principle be applicable cross-linguistically, leaves the problem fairly unconstrained. We refine these constraints in ways motivated by the nature of the grammar (the ERG) and the language for which it is designed (English) to create a tractable subtask.

We focus here on paths linking nominal predicates, since intuitively they seem likely to benefit from such an approach. We aim to identify inference rules relating two DMRS subgraphs G_1 and G_2 to each other, where both contain identical nominal lexical predicates s and e.[2] We refer to s and e as *anchor predicates*, since they are required to be the same in the inference rules. The remainder of the nodes in G_1 and G_2 must be light predicates. We aim to learn a set of such abstract mappings which relate graphs containing these pairs of anchor predicates to each other. Because of the way nominal predicates are treated in DMRSs produced by the ERG, we are interested specifically in paths containing the anchor predicates, a path linking those predicates, and the quantifier for the anchors.

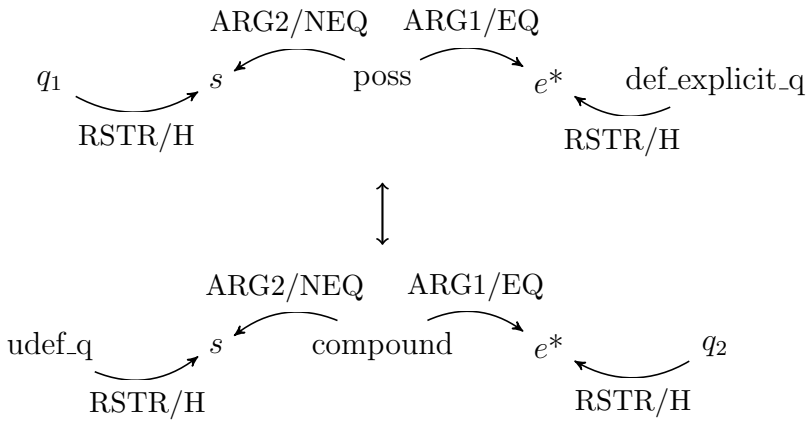


Figure 7.2: A pair of abstract DMRS subgraphs showing a mapping between a possessive and a noun compound construction. '*' denotes the phrasal head.

Figure 7.2 illustrates this, showing a mapping between the possessive and noun compound constructions, which is an example of the kind of mapping we would like to automatically acquire. In the figure, s and e are the anchor predicates, which are placeholders for nominal predicates and which would correspond to specific predicates in the concrete DMRS to which the mapping could applied (as distinct from the abstract subgraph we show here). Similarly, q_1 and q_2 are placeholders for quantifiers. The subgraphs each specify particular quantifiers as well, but it is the quantifier of the phrasal head which is specified for the possessive, and the quantifier of the non-head specified for the compound constructions. We could in principle allow

[2]If we were using this technique to assign DMRSs a loose similarity score, we could allow s and e to be different as well, for example by allowing synonyms, but this is beyond the scope of this chapter.

for different possible combinations of quantifiers on each endpoint. However, as we discuss in Section 7.4, the mappings are acquired from parsed corpus data. This leads to sparse data problems if we attempt to enumerate all possible combinations of quantifiers, so we use limited information from the type hierarchy of the grammar to make generalisations, as we discuss further in Section 7.5.1.

7.4 Learning DMRS path correspondences

Our methodology for acquiring DMRS mappings is inspired by the work of Lin and Pantel (2001) described in Section 7.2. We assume a version of their 'Extended Distributional Hypothesis' such that if we see a particular pair of nominal predicates (e.g., `_synthesis_n_of` and `_aspirin_n_1`) related by graphs G_1 and G_2, then we take this as evidence that a mapping between G_1 and G_2 is plausible. In fact, since we are only considering the light predicates as candidates for inclusion in the subgraph, we are using a fairly constrained version of this hypothesis — many possible subgraphs will not be considered. We make the simplifying assumption that every occurrence of a proper or common noun predicate in our text corresponds to a similar entity. The most obvious violation of this is word-sense ambiguity that is not captured in the ERG, but it is a reasonable approximation to make, as most words, particularly common ones, have an overwhelmingly dominant sense accounting for most usages (Kilgarriff 2004; Koeling *et al.* 2005).

We thus look for subgraphs relating nominals in a training corpus, and record the subgraphs in abstracted form, avoiding information specific to the DMRS, as described in more detail below. If we see a particular pair of abstract subgraphs G_1 and G_2 many times in a training corpus, relating the same noun endpoint predicates S_1 and E_1 each time, that provides much stronger evidence that the substructures are related. We are unlikely to see these two particular nouns in connected structures a large number of times even in a very large corpus (and if we did, we might be suspicious of non-decomposable multiword expressions being the source of this — see e.g. Baldwin *et al.* (2003)). However by keeping track of each different endpoint pair (S_1, E_1), (S_2, E_2) etc. observed with the abstract subgraphs of interest G_1 and G_2, we can build up a large amount of evidence for the strength of the relationship between G_1 and G_2. By comparing all attested substructures in this way and scoring the correspondences on this basis, we are then able to build a set of these correspondences and their strengths.

We take as input a training corpus parsed with the ERG, with each sentence output in DMRS format. We find all subgraphs within the DMRSs connecting two nominals. We then iterate through these subgraphs, and if a subgraph satisfies certain conditions (described in more detail below), we keep a record of the predicates corresponding to the internal nodes of the subgraphs and the DMRS links that connect them, alongside the nominal endpoints. We thus obtain abstracted DMRS subgraphs

Abstract Path	(**X**, **Y**) Counts
$\mathbf{X} \xleftarrow[\text{NEQ}]{\text{ARG2}} \text{compound} \xrightarrow[\text{EQ}]{\text{ARG1}} \mathbf{Y}$	(_synthesis_n_of, _aspirin_n_1): 1
$\mathbf{X} \xleftarrow[\text{NEQ}]{\text{ARG2}} \text{\_for\_p} \xrightarrow[\text{EQ}]{\text{ARG1}} \mathbf{Y}$	(_synthesis_n_of, _aspirin_n_1): 1

Table 7.1: Abstract paths and counts that would be obtained from training on sentences including the subgraphs shown in Figures 7.1(c) and 7.1(d)

which we can combine into a tuple $\langle G_1, G_2, s \rangle$,[3] for the source subgraph G_1, the target subgraph G_2, and the correspondence score s between them. As implemented here, the rules are always symmetric between G_1 and G_2 — i.e. there is always a reversed version $\langle G_2, G_1, s \rangle$ with the same score, but depending on how the score was calculated, it would be possible to have asymmetric rules.

DMRS does not have an intrinsic notion of ordering, so each link between endpoints gives two subgraphs — in one arbitrary direction, and in the opposite direction. However, the apparent differences between these are spurious, so the abstract subgraphs are normalised to a standard ordering based on the labels of the graph links to make such links appear identical, and duplicates are subsequently removed.

To give a concrete example of the whole process, let us assume we have a training corpus with full sentences which include the fragments shown above in Figures 7.1(c) and 7.1(d). From these fragments, we would see that the predicates `_synthesis_n_of` and `_aspirin_n_1` occur in both sentences. We would find the paths joining the predicates, and if these conform to the requirements set for such paths, we would then normalise the directionality on the abstract subgraph to remove the duplicate reversed versions as described above. The endpoint which comes first after this arbitrary sorting is considered the first endpoint (labelled **X** here), and the other endpoint is considered to be the last (labelled **Y**). This would give the counts shown in Table 7.1.

Thus we would posit a relationship between the only two abstract subgraphs in the table, since they both have one matching endpoint pair, giving the rule in (7.5) as well as the version where G_1 and G_2 are swapped.

(7.5)

$$\left\langle \begin{array}{c} \mathbf{X} \xleftarrow[\text{NEQ}]{\text{ARG2}} \text{\_for\_p} \xrightarrow[\text{EQ}]{\text{ARG1}} \mathbf{Y} \\ \mathbf{X} \xleftarrow[\text{NEQ}]{\text{ARG2}} \text{compound} \xrightarrow[\text{EQ}]{\text{ARG1}} \mathbf{Y} \end{array} \right\rangle$$

[3] We sometimes display this vertically and omit the score for readability.

7.4.1 Conditions on Subgraphs

When examining DMRSs for candidate subgraphs connecting anchor points, we place certain conditions on the subgraphs and the anchor points to constrain the candidates to those which are likely to be most useful. The nodes in the connecting subgraph are required to be light predicates and we limit the search to endpoints connected by at most 4 intermediate nodes.

In addition, as mentioned in the previous section, we only examine connections between nominal nodes in the DMRS.[4] More specifically, we require that each endpoint node either has its part-of-speech listed as 'n' in the DMRS predicate, which in our parsing configuration indicates the predicate corresponds to a common noun, or that the predicate is '`named_rel`', which is primarily used for proper nouns.

When these proper nouns occur, it is not clear what the optimal strategy to handle them should be. Depending on the nature of the source text, it could be the case that many of the names mentioned are roughly equivalent as anchor text endpoints, so in this case our algorithm should reflect this by treating the different proper name instances as equivalent. On the other hand, it is possible that there are also important differences between the various names, particularly between different classes of entities such as people's names compared to locations or organisations. In the latter case we may wish to keep more separation between the various proper name instances. For example, if we have the sentences *Mozart was born in Salzburg* and *Salzburg was the birthplace of Mozart*, we would want to treat the occurrences of *Mozart* as the same endpoints, and keep these distinct from *Salzburg*. We report results for two strategies, representing opposite extremes on the continuum of separation of `named_rel` instances.

- ConflateNE: In this strategy we can conflate all `named_rel` endpoints, so that any proper name is equated with any other. In this case, the respective DMRSs corresponding to *the* ***Smith*** *method* and *the* ***Jones*** *method* would both appear identical for the purposes of extracting anchor endpoints.

- SplitNE: In this case, we split the instances apart by source string, so that *Smith* would only match with another identical string as the endpoint.

Another option would be some kind of middle ground, applying a named-entity tagger to classify the entities as particular types, and only group together similar entities, but this is not something we have investigated further.

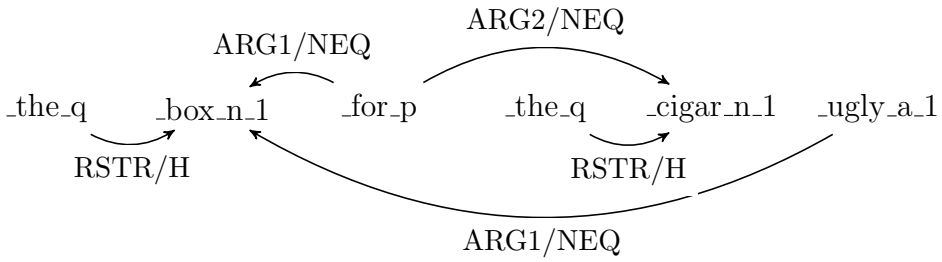


Figure 7.3: A test DMRS for *The box for the cigars is ugly*

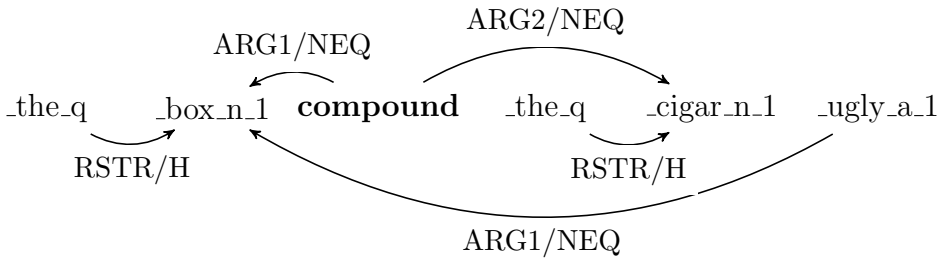


Figure 7.4: The DMRS in Figure 7.3 after remapping using rule (7.5)

7.5 Applying learnt correspondences

We wish to make use of correspondences between abstract graphs on an unseen DMRS X. Here, we can use a similar method to the learning stage to extract link paths with appropriate conditions on the endpoints and internal paths. If a link path would be valid for rule learning, it is also possible that it will correspond to a path in the matrix of matching DMRS subgraphs. Checking this can be achieved in the obvious fashion — we simply strip off the endpoints and create an abstracted subgraph in the same way as in the rule learning stage. If this matches any abstracted path for which we have learnt correspondence rules, all rules with a score above a particular threshold would be output as potential semantic equivalents. We return to the question of how these scores can be calculated in Section 7.5.3.

In a real-world system, for example question answering or information extraction, we would then use these roughly equivalent corresponding subpaths in, for example, DMRS matching. If we were comparing X to a second DMRS Y and had learnt a rule $\langle G_1, G_2, s \rangle$, we could boost the matching score if X had a subgraph corresponding to

[4]This is a slight simplification — as discussed in Section 7.5.1 we also look at certain links not directly connecting nominals in particular cases.

abstract subgraph G_1 and Y could be matched to G_2, if the concrete subgraphs have noun endpoints that are similar according to some metric.

Here we evaluate our rules by applying the postulated mapping to X so that we can investigate the correctness of the MRS-to-MRS mapping and generate paraphrases. This is mostly straightforward. Assume there is a matching concrete subgraph C in X, which has abstracted path G_1. Applying rule $\langle G_1, G_2, s \rangle$, we can delete all nodes and links in C corresponding to predicates and links in G_1, and create new nodes and links in C corresponding to the predicates and links of G_2. The new DMRS can then be converted to an MRS which is then used for generation with the ERG within the LKB, as described in Section 3.5.5.

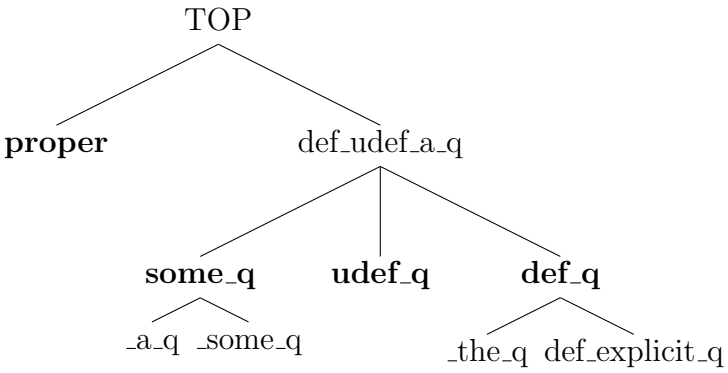
For example, if we had learnt from the training data the ruleset of just two rules (Rule (7.5) and its reverse) described in the example in Section 7.4, and then observed the test DMRS shown in Fig 7.3, we would match the DMRS to the left hand subgraph in the version shown, and transform the DMRS to the version shown in Fig 7.4, where the remapped predicates and links are highlighted. In this particular example, it happens to be the case that the link annotations do not change, but this will not generally be so. The reader may suspect that the DMRS in 7.4 is slightly ill-formed, since nouns in compounds are generally not outscoped by instances of quantifiers such as *the*, which is what the DMRS implies. This is indeed the case, and we return to this problem in the following subsection.

7.5.1 Handling Quantifiers and Noun Number

As discussed in Section 7.3.3, quantifiers need special treatment. Since one of our evaluations is the ability to generate from the converted semantic representations, an error in assigning the quantifier will result in an otherwise well-formed semantic representation failing to produce output in the generator. At the same time, if we are too specific about the attached quantifiers in rule-learning or application, we risk missing interesting generalisations.

We make use of a subset of the quantifier hierarchy found in the grammar we used to parse the training and test data. We manually specify a small number of privileged quantifier predicates that are allowed to appear in rules, but apart from this we only make use of the information already available in the ERG. Each potential endpoint noun will always have a quantifier attached to it in the DMRS. When learning rules, if a quantifier is not from the privileged list, we back off up the hierarchy until we find a quantifier that is. The same procedure applies when finding paths from test DMRSs to remap, so matching a path to a known rule is trivial. The quantifiers are not on the direct dependency path linking the nouns, so could be paralleled with the "satellite links" used by Snow *et al.* (2005) for relation extraction, which also fall outside the dependency path.

In the rule application stage, however, we only modify quantifiers in the remapped version of the DMRS if the original quantifier is incompatible with the quantifier in



(a) EXTENDED Hierarchy

(b) UDEFORTOP Hierarchy

Figure 7.5: The quantifier hierarchies we use

the new DMRS path. That is, there might be a test DMRS X with subgraph C which has a left-hand endpoint e_1 with quantifier predicate q_1. The abstract link path corresponding to this is G_1, and we have a rule $\langle G_1, G_2, s \rangle$, where G_2 stipulates quantifier q_2 for the left endpoint. In this situation we would only replace the quantifier predicate in the output DMRS with q_2 if q_1 was not subsumed by q_2 in the hierarchy. Otherwise, the node outscoping the left-hand endpoint with q_1 as the predicate would remain unchanged in the output DMRS. The idea is that if the source DMRS already has a quantifier predicate that is consistent with the output, there is no need to change it, but if the rule demands an incompatible predicate, it is necessary to change it. We have two different hierarchies that we evaluate for rule generation and application, shown in Fig 7.5, with acceptable quantifiers shown in bold.

To illustrate this, suppose that in our extended definition of the abstract subgraphs, the example rules we have learnt take the following form:

(7.6)

$$\left\langle \begin{array}{l} \text{def\_q} \xrightarrow[\text{H}]{\text{RSTR}} \mathbf{X} \xleftarrow[\text{NEQ}]{\text{ARG2}} \text{\_for\_p} \xrightarrow[\text{EQ}]{\text{ARG1}} \mathbf{Y} \xleftarrow[\text{H}]{\text{RSTR}} \text{def\_q} \\ \text{def\_q} \xrightarrow[\text{H}]{\text{RSTR}} \mathbf{X} \xleftarrow[\text{NEQ}]{\text{ARG2}} \text{compound} \xrightarrow[\text{EQ}]{\text{ARG1}} \mathbf{Y} \xleftarrow[\text{H}]{\text{RSTR}} \text{udef\_q} \end{array} \right\rangle$$

If we were then attempting to apply mappings to the DMRS in Fig 7.6 (a duplicate of Figure 7.3 with boxed labels added as reference points) the rule would match, as the source DMRS has exactly corresponding matching subgraphs after backing off

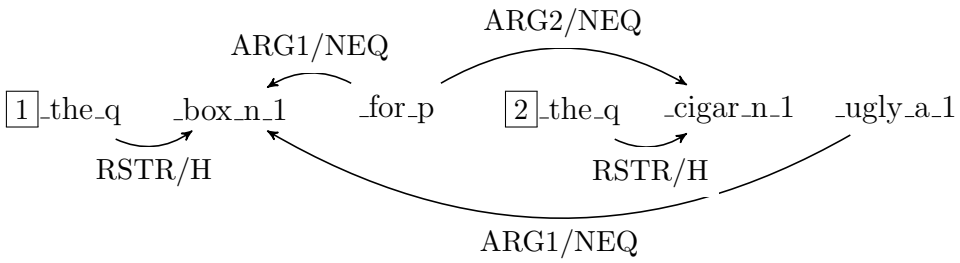


Figure 7.6: A test DMRS for *The box for the cigars is ugly*, repeated from Figure 7.3, but with boxed labels added for reference within the text.

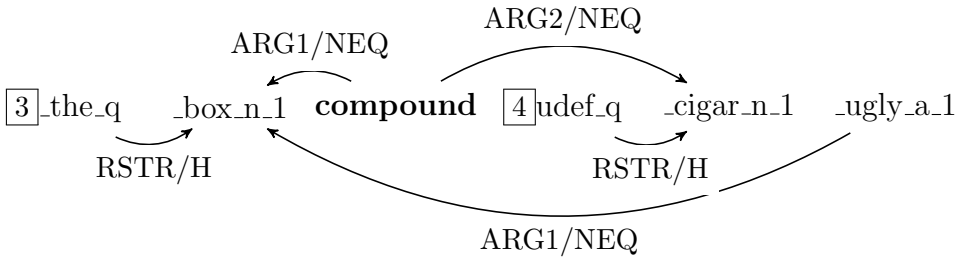


Figure 7.7: A test DMRS after remapping, similar to that in Figure 7.4 (with the addition of boxed labels) after a quantifier remapping stage to replace the predicate at 4.

up the quantifier hierarchy as described. In the target abstract subgraph of the rule, the left-hand quantifier is already `def_q`, which subsumes the backed-off observed quantifier `def_q`, so the corresponding DMRS predicate `_the_q` (1 in the figure) remains unchanged (as 3). The other `_the_q` (2) also backs off to `def_q`, which is not subsumed by `udef_q` in the hierarchy, so we would replace it with `udef_q`, instead of `the_q` at 4, as shown in Figure 7.7 (c.f. Figure 7.4 where the mapping has not occurred). The remainder of the predicate remapping is as described above. The resultant Figure 7.7 DMRS would generate the sentence *The cigar box is ugly*, as we would hope. Without this quantifier handling, the output DMRS would not generate output at all.

Additionally, when generating rules, we only consider endpoints where the quantifier is from a particular whitelist – specifically `udef_q`, `_the_q`, `_some_q` and `_a_q`, to avoid the risk of noise from allowing a wider range of more marked predicates. Finally, to enable broader generation from the resultant re-mapped DMRS, we also delete the attributes from the endpoint nodes corresponding to number and countability, leaving them underspecified. This allows the generator to determine which out of the singular and plural variants are licensed by the semantics of the other parts of the sentence.

7.5.2 Selecting Corpora

The data-driven approach we are taking could, in principle, learn rules from any parsed corpus where the output has been converted into DMRS. The domain(s) of the corpus should not make a particular difference, except to the extent that a particular syntactic structure could have a vastly different distribution in a particular domain.

Nonetheless, certain corpora will be more appropriate than others. In particular, having the text focussed on a certain small domain may be advantageous. Firstly a small domain means that we are likely to get more matching noun pairs, since we would expect certain salient nouns to occur relatively frequently. Secondly, the potential problem of sense ambiguity in the target nouns is likely to be alleviated. However, for the purposes of extracting a reasonable number of correspondences, and to minimise the effects of noise in the data, it is desirable for the corpus to be relatively large.

It is not essential that the parsed corpus is entirely accurate, since we would hope the volume of data would override the effects of some incorrect parses. Nonetheless, a parsed corpus with human-selected parse trees should make development cleaner and possibly enable us to learn more from a smaller amount of data.

With these factors in mind, we selected two corpora to use as training data. Firstly, we selected the WeScience corpus introduced in Section 3.4.1 and used in most chapters of this thesis. It fulfils all of these criteria: it is for a single domain, yet with close to 9000 sentences, it is also large enough to learn meaningful relationships. Additionally, it has gold-standard parses available so we can investigate the effect of

the data-set accuracy on learning these correspondences, by comparing with DMRSs from automatically-created parses. As well as this, we also make use of the LOGON Treebank which we introduced in Section 3.4.1. It has many similar characteristics to the WeScience data: it is in a single domain and has gold-standard parse trees available for it. Having these two separate corpora enables us to investigate the effects of the domain on the rules learnt during training.

7.5.3 Ranking Alternatives

We wish to place a higher weighting on related paths in which we have more confidence, based on some function of the number of overlapping noun endpoint matches between the two paths, and a lower weighting on those that are less plausible from the data, with some threshold below which the correspondence can be ignored. The approaches we experiment with for this are:

- NAIVE: The similarity of a given pairing of paths is simply the number of noun endpoints that they have been observed with that overlap between two pairings. This is globally scaled so that the maximum similarity between any paths is 1.

- IDF: This attempts to improve on the simplistic method above by placing more emphasis on overlaps between rarer nouns, since, as noted in Lin and Pantel (2001), these are likely to be more significant. Here, we use the *inverse document frequency* metric (Manning *et al.* 2008: Chapter 6) from information retrieval, treating each training sentence as a document. The raw overlap counts are multiplied by the inverse document frequency of each term in the overlap pair — specifically the version:

  $$I = \log\left(\frac{N}{d_t}\right) \tag{7.7}$$

  where N is the number of documents (sentences), and d_t is the number of documents the term t (here actually a noun predicate) appears in. Again, the similarities are globally scaled to $(0, 1)$.

- PAIRIDF: A variant on IDF attempts to take into account the fact that it is not just individual term frequencies that matter, but also the frequency of the pairing. If a given noun pair occurs in the corpus with two different paths, and the pairing of nouns is very unlikely in the corpus, we should give more weighting to the correspondence than if it were a very common pairing of nouns co-occurring in both situations. With PAIRIDF, we also use Equation (7.7), but use each observed pairing of noun endpoints as the term t, instead of having t refer to a single endpoint. There is an obvious potential data-sparseness problem here – since we are looking at pairings, a large proportion of them will

be singletons, so we cannot give too much credibility to the counts we obtain unless the corpus is very large.[5]

- JACCARD: The well known Jaccard coefficient from information retrieval (e.g. Manning *et al.* (2008)) of sets A and B is $|A \cap B|/|A \cup B|$. If we define A and B as the set of noun endpoints that are observed with two abstracted potentially corresponding DMRS paths, the Jaccard coefficient can be used a measure of the similarity of the two. This has the desirable property that a path has a similarity of exactly 1 with itself, and in other cases the scaling to (0, 1) happens naturally.

7.6 Evaluation

7.6.1 Generation as a Stand-In

An ideal evaluation here would be task-based. Any task involving (D)MRS comparison, such as Question Answering, Machine Translation or DMRS-based querying (for example, the facility provided by SearchBench (Schäfer *et al.* 2011)) could conceivably benefit from this, as we could potentially broaden the range of matched DMRSs in order to boost recall. Since such a system with the required annotated data is not immediately available,[6] we show evidence for the plausibility of the DMRS alternatives. We first measure how many of the rules learnt are applicable to a separate test corpus, which is an indicator of the utility of a rule — if it matches real-world sentences, it may have some application in an external system. Secondly, we measure the ability of an applied rule to produce new DMRSs from which we can generate, which we denote *generatable*. If a rule is generatable, there is a strong argument for its well-formedness (although the converse is not necessarily true).

For the test corpus, we parsed every 1000th sentence of the British National Corpus and removed sentences of length 12 or more for tractability in generation, giving 1982 sentences. We primarily trained on the WeScience corpus, using the hand-selected parse trees, but to investigate the effects of the corpus on rules learnt, we also trained on the LOGON corpus (also with hand-selected parse trees). We were also interested in the question of how important the parse quality of the training corpus is, so in one set of experiments, instead of using the gold-standard parses to produce DMRS, the training data comes from the best WESCIENCE parses according to an out-of-domain (LOGON) parse selection model.

[5] One way around this without needing to parse prohibitively large quantities of data might be to get the IDF counts from a very large unparsed corpus, although it would ideally be the same domain as the parsed training corpus, which is difficult to obtain, and not something we have investigated further.

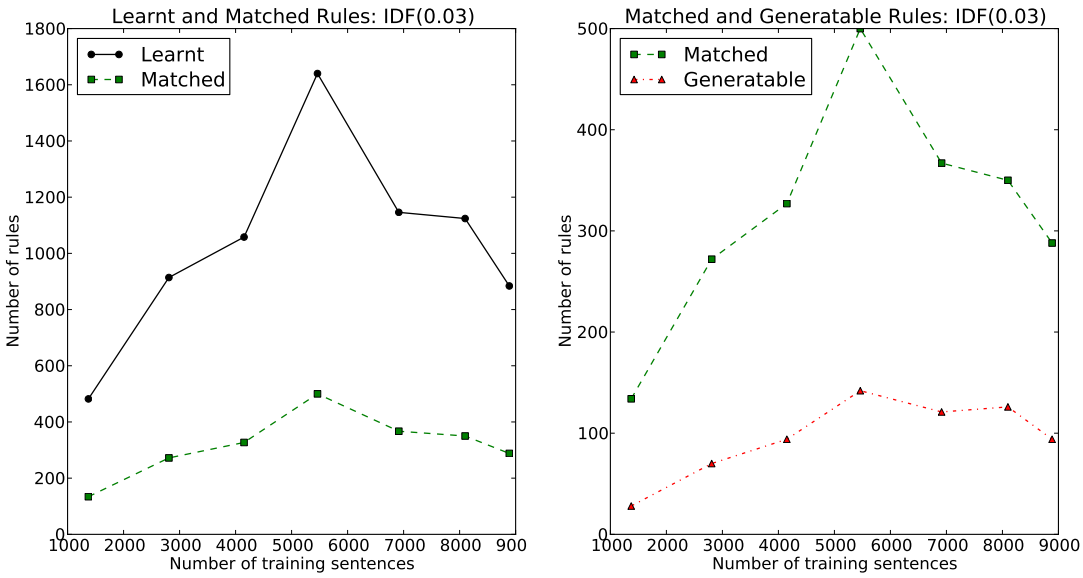
[6] While the SearchBench DMRSs are available, we have no annotated data set to check for a performance improvement.

Train Corp	Quant Hier	NE Strat	Number of Rules		
			Learnt	Matched	Gen'd
WeSc	UdefOrTop	ConflateNE	852	292	84(9.9%)
WeSc	Extended	SplitNE	960	380	110(11.5%)
WeSc	UdefOrTop	SplitNE	898	387	103(11.5%)
WeSc	Extended	ConflateNE	884	288	94(10.6%)
WeSc(alt)	Extended	ConflateNE	602	203	64(10.6%)
LOG	Extended	ConflateNE	5918	977	297(5.0%)

Table 7.2: Rules generated and applied to BNC test corpus with IDF ranking and a threshold of 0.03, over various training corpora and configurations. WeSc(alt) uses automatically-parsed sentences; all other items use gold parsed sentences. Percentages in "Gen'd" column indicate what percentage of learnt rules were generatable.

Ranking	Thresh	Number of Rules		
		Learnt	Matched	Gen'd
Naive	0.040	1816	448	68(3.7%)
Naive	0.050	696	161	29(4.2%)
Naive	0.060	696	161	29(4.2%)
Naive	0.070	426	98	22(5.2%)
IDF	0.030	884	288	94(10.6%)
IDF	0.040	496	177	60(12.1%)
IDF	0.050	240	85	33(13.8%)
IDF	0.060	176	63	26(14.8%)
Jaccard	0.004	9614	1212	46(0.5%)
Jaccard	0.006	6619	1184	37(0.6%)
Jaccard	0.008	5911	1159	32(0.5%)
Jaccard	0.010	4765	1150	31(0.7%)
PairIDF	0.020	4696	1171	233(5.0%)
PairIDF	0.040	874	250	66(7.6%)
PairIDF	0.060	406	109	29(7.1%)
PairIDF	0.080	260	71	21(8.1%)

Table 7.3: Rules generated and applied to BNC test corpus with Extended quantifier hierarchy, and ConflateNE strategy for handling `named_rel` instances



(a) Learnt and matched rules

(b) Matched and generatable rules

Figure 7.8: Learning curve created by successively adding sections of the WeScience corpus, using Extended quantifier hierarchy and ConflateNE, with IDF ranking and a threshold of 0.03

For ranking of rules, for most experiments we selected the IDF method with a threshold of 0.03 required, since initial experimentation showed this to be a reasonably solid performer in at least some configurations. We experimented with various combinations of rule-learning parameters. Firstly, we evaluated both quantifier hierarchies (Extended and UdefOrTop) from Section 7.5.1. Another dimension of variation was the two strategies (SplitNE and ConflateNE) described in Section 7.4.1 for handling `named_rel` predicates. We show the results under these different conditions in Table 7.2.

Comparing the various rows, it seems that the choice between Extended and UdefOrTop is not particularly important, as the difference between one and the other is 10 or less in terms of the numbers of generatable rules created (an absolute change of 12%), and 62 or less comparing the number of rules learnt in total (an absolute change of 7%). Changing from ConflateNE to SplitNE has a slightly larger effect on the number of generatable rules, with 16–19 (17–22%) more created, but a smaller effect (40 or less) on the number of rules learnt. We arbitrarily selected Extended and ConflateNE for the remaining experiments; SplitNE achieved a

slightly higher percentage of generatable rules for the IDF ranking method, but the differences were fairly small.

Table 7.2 also includes results for using different training corpora. These had a larger effect than the other training parameters (although it is possible that some of these effects are simply due to the arbitrarily-selected thresholds, which should possibly be tuned differently for the different corpora). Using automatically-created WeScience parses instead of hand-selected ones results in approximately a 30% drop in the number of learnt and generatable rules. Switching to LOGON, however, had a much larger effect, with a sixfold increase in the number of rules learnt, and a threefold increase in the number of generatable rules. For subsequent experiments, we use WeScience, since the 'quality' of rules seems higher, in terms of the percentage which are generatable.

Having chosen a semi-arbitrary set of training parameters, we also wished to investigate whether the choice of ranking method and threshold made a large difference. In Table 7.3, we show the effect of holding the corpus constant and applying various combinations of ranking methods and thresholds. The choice of ranking method makes a large difference in terms of the percentage of learnt rules which are generatable. Jaccard is the least successful in these terms — to obtain more than a few generatable rules, it is necessary to set the threshold so high that many thousands are learnt, but most of these are not shown to be usefully applicable to new sentences, indicating that the ranking method is probably suboptimal. Naive and PairIDF are both intermediate in terms of the quality of rules created, with a slightly higher percentage of generatable rules coming from the more informed PairIDF method. Meanwhile, the most effective method according to these results was IDF. Reassuringly, increasing the required threshold in all cases results in a higher percentage of rules which generate, however this increase is less pronounced when the ranking method is producing a lower percentage of generatable rules in the first place, further indicating that these methods are not successful at ranking the rules in terms of the most likely candidates.

Finally, it is also interesting to consider the effect of corpus size while holding all other training parameters constant, to create a learning curve. We used the standard parameters from above — Extended quantifier hierarchy, IDF ranking method thresholded at 0.03 and WeScience training data, with subsets created by concatenating successively more subsections from the original sections of the WeScience corpus.[7] In Figure 7.8, we show the results of this, for the number of rules learnt, the number of rules successfully matched, and the number of generatable rules over various sizes of training corpus. The curves do not increase monotonically (and indeed there is no reason why this needs to be the case), but as more sentences are added, the percentage of generatable sentences increases slightly, suggesting the rules may

[7] This is in contrast to the shuffling we used through Chapter 4 for parse selection learning curves.

be of higher quality with more training data. On the other hand, it is still possible to obtain respectable quality from only around 3000 training sentences.

7.6.2 Examples of Generation

To illustrate the possibilities of generation, we show some hand-selected generated sentences in (7.8)–(7.10), as well as the originals in boldface. We also show the scores obtained using IDF. There are unsurprisingly many less appropriate rephrasings (the quality of the outputs can also be damaged by parsing errors in the test sentences); we have not evaluated how they are split between good quality and poor quality paraphrasings.

(7.8)
ORIG: ***The authors state that citation counts indicate impact rather than quality.***
0.281: *The authors state the counts of citations indicate impact rather than quality.*
0.236: *The authors state the count of the citations indicates impact rather than quality.*
0.210: *The authors state the count of some citation indicates impact rather than quality.*
0.171: *The authors state that counts of citations indicate impact rather than quality.*
0.108: *The authors state some counts of citations indicate impact rather than quality.*

(7.9)
ORIG: ***Doc Threadneedle leaned over and kissed her.***
0.047: *Threadneedle, a doc, leaned over and kissed her.*
0.047: *Threadneedle, some doc, leaned over and kissed her.*
0.047: *Threadneedle, some docs, leaned over and kissed her.*

(7.10)
ORIG: ***He failed a university entrance exam.***
0.054: *He failed an exam of university entrances.*
0.046: *He failed an exam for university entrances.*
0.041: *He failed exams such as university entrances.*
0.041: *He failed the exam of the university entrance*
0.041: *He failed the exams of the university entrances.*

7.6.3 Discussion

The results we have found here for this exploratory investigation are somewhat mixed. On the one hand, we were able to learn a large number of inference rules, but it is of course not entirely clear how useful they are. In the various configurations shown in Tables 7.2 and 7.3, generally around 25–35% of the learnt rules were able to

be matched to DMRSs in a new corpus in a different domain (with a few exceptions, particularly using JACCARD). This is a sizeable proportion, although it is not clear why the majority were never able to be reapplied successfully. There is a similar "success rate" in going from rules which were matched to those which were able to be generated from, although in this case JACCARD was an obvious exception, with under 2.5% of learnt rules able to generate in all cases.

Overall, there were generally respectable success rates, with 4–15% of rules learnt able to be applied to new sentences and produce DMRSs which were well-formed enough to be able to produce new variant sentences. The only exception was the JACCARD ranking method at 0.8% or less, which we discuss more later. Regardless, this obviously leaves much room for improvement — in all cases, the vast majority (75–99%) of rules which could be derived from the training data were unable to be successfully applied in a way which could create DMRSs suitable for generation, despite various efforts we made to create DMRSs suitable for generation as discussed in Section 7.5.1.

There are several possible reasons for the non-viability of many of the DMRSs we create. The DMRSs could of course simply be ill-formed, due to an imperfect rule application process. But it is also possible that there are bugs in the semantics produced by the ERG — Flickinger *et al.* (2005a) found that 8–11% of sentences in two corpora had clearly ill-formed semantics, with a further 11–17% having semantics which was suspect according to a proposed test. In particular, particular grammar rules were systematically producing ill-formed semantics, so if our algorithm was learning rules which corresponded to such invalid semantic structures, they could never be expected to produce output from the generator. Additionally, White (2011) notes that many systems for chart-based language generation suffer from incomplete coverage and proposes "glue rules" as a way to overcome this problem. The glue rules allow the creation of fragment edges in the chart, which can be used in subsequent chart edges with relatively little modification to the realisation process. This enables the chart realiser to find a derivation where it would not otherwise be able to, in some ways emulating dependency realisation, which is reliably more robust. A similar approach could possibly be applicable here.

In terms of the importance of selecting training parameters, we found some other interesting results. From Table 7.2, we can see that the choice of strategies for handling quantifiers and proper nouns is not particularly important when we use the IDF ranking method — all produce similar quantities of rules matched. However, the training corpus has some effect: using automatically-parsed sentences causes a small drop in the number of rules learnt (although the same overall success rate). Using LOGON instead makes a large difference in the number of rules learnt, although this difference is likely to be largely the result of inappropriate thresholds which chosen on the basis of the WESCIENCE corpus.

The various methods for ranking rules based on endpoint counts made a fairly large difference, as we can see from Table 7.3. The NAIVE method was surprisingly effective

on its own, with around 5% of rules postulated having strong evidence for their validity in being able to generate new sentences, while the PairIDF produced a slightly higher success rate. The IDF method was the most successful overall, however there was (perhaps unsurprisingly) a large degree of overlap between IDF and PairIDF. With a threshold of 0.04 for both methods, of the 177 and 250 rules matched by the IDF and PairIDF individually, 141 of these matched for both ranking methods; similarly, from the 60 and 66 respective generatable rules, 50 appeared in the results from both methods. Jaccard stood out as a very poor ranking method for selecting rules likely to be amenable to generation; the reasons for this are not clear.

The effect of changing the training set size while holding other parameters constant also produced interesting results in Figure 7.8. There is a strong peak, indicating that one particular section was able to provide a large number of rules, and this persisted through the pipeline, in that a large number of the rules were generatable. Possibly there was some characteristic of these first few sections, such as a large number of rephrasings including similar noun endpoints, which contributed to this. However, this effect was counteracted as new sections were added, with the number of rules learnt decreasing.

Naively, we may expect the number of rules exceeding the threshold to monotonically increase as we increase the number of documents. The N denominator in Equation (7.7) will increase, resulting in a higher IDF score and hence score for rules against the threshold if all other aspects are equal and we keep the threshold the same. However, other aspects are not equal — the value of the d_t denominator may increase in greater proportion than N, and the raw counts (which we multiply by the IDF value) for the relevant terms may decrease in relative terms. Regardless, one interesting tentative conclusion is that not all training corpora are necessarily equal for learning these inference rules — sometimes a subset produces more high-quality generatable rules. However, more experimentation would be required to determine if this is simply an artefact of scaling effects while we were using a somewhat arbitrary fixed threshold.

Overall, we were able to reliably produce several hundred rules able to be matched to DMRSs from a new domain, of which a significant proportion of 25–35% could produce new well-formed DMRSs. In each configuration we have strong evidence that at least a few dozen rules are useful, and the selected examples of generation we have shown also support this.

It is probably apparent to the reader that the work discussed in this chapter is relatively immature. We have attempted to show that the techniques presented, or a generalisation of them, is at least plausibly useful. Much more work could be done to more fully investigate the capabilities and limits of this approach. In particular, as we have already emphasised, a more realistic evaluation would be highly desirable. We discuss more possibilities for extending this work in Section 8.5.1.

7.7 Summary

In this chapter we investigated the problem of abstracting away from superficial semantic differences in the DMRS output of grammars such as the ERG. These deep representations frequently produce different structures for particular sentences which would be interpreted human readers as having different semantics, and this problem is most obviously evident in noun phrases to which we devoted all of our attention. Shallower techniques could successfully equate some of these superficially different phrases, and thus obtain higher recall, but also risk losing a large amount of precision in false positives. Here we aimed to preserve the precision we expect in a deep parsing approach but boost recall when matching DMRSs, by learning robust DMRS inference rules for matching up different noun phrases.

We learnt these rules by using DMRSs derived from parsed corpora, and the notion of *anchor text*: we assumed that if the same nominal endpoint pairs frequently occurred joined by different DMRS paths, there was evidence for a relationship between the paths. By building up counts of endpoints and paths across a large number of sentences (with special handling of particular aspects such as quantifiers), we learnt inference rules based on the DMRS path correspondences and used a ranking heuristic to assign a confidence score to the rule.

Ideally we would have used these confidence scores to weight possibly-related DMRSs, but lacking a test corpus, we instead presented an evaluation based on tactical generation. We applied a cutoff threshold to the confidence scores to include or exclude rules, giving between hundreds and thousands of rules. Using the most successful metrics we could generate from between 10–15% of these. We were generally able to produce a few dozen rules which could generate new sentences, providing strong evidence that the inference rule is a plausible candidate to generate semantic equivalences. While there could be much refinement of the technique and more convincing results would be desirable, we have demonstrated the promise of the approach.

Part IV

Summary

Chapter 8

Conclusions and Future Work

8.1 Introduction

In this thesis, we have presented an investigation into deep parsing using the English Resource Grammar, investigating ways to improve parsing accuracy, as well as ways in which the semantics of the grammar can be used as input to other tasks. We intended to make deep parsing a more useful tool, as well as showing its value for solving downstream tasks.

In this chapter we summarise each of the content chapters of the thesis in a separate section. Chapters 4 and 5 from Part II, focussed on improving parse selection accuracy, are covered in Sections 8.2 and 8.3. Chapters 6 and 7 come from the application-oriented Part III, and are covered in Sections 8.4 and 8.5.

Alongside each summary, we provide a "Discussion and Future Work" section. As well as indicating future directions in which the research could be taken, we give a critical analysis of the research which we performed during the chapter. Research will inevitably have flaws even in the best case, and much of the research on which these chapters were based took place in the early stages of the thesis. With the benefit of hindsight, it is possible to identify flaws in much of this work, but with limited time, it is not possible to correct all of them. The discussion in these sections is intended to critically analyse the research we conducted in the earlier stages, and indicate that these flaws have at least been recognised, even if they have not been rectified.

8.2 Domain Adaptation

In Chapter 4, we quantified the impact of training domain on parse selection accuracy when using the English Resource Grammar. The ERG, in contrast to statistical treebank-derived grammars on which much previous work has been based, is a large set of manually-specified grammar rules and lexical entries, and the statistical model is exclusively used to choose between the parse trees licensed by the grammar for a given

string of text. It was not clear how these differences compared to previously-studied treebank parsers would affect the cross-domain performance penalty observable with the ERG.

We evaluated the domain adaptation performance penalty using the LOGON and WeScience corpora, two large ERG treebanks. We used these as training corpora, with held-out sections of each as test sets, along with two extra test corpora. We found that using only an out-of-domain training corpus did indeed cause a drop in parsing scores compared to using only an in-domain corpus. The penalty was 11–16% for Acc_1 and Acc_{10}, the stricter exact tree match metrics, and 5.5–6% for the dependency-based EDM_{NA} metric. The dependency scores are not greatly different from those found in other work, although the domains in question are of course different.

We also evaluated how the parse selection scores changed as the volume of training data was increased, and compared the relative value of in-domain and out-of-domain training data, finding that in-domain training data has considerably more value, with as few as 10,000 in-domain training tokens providing better accuracy than 90,000 tokens of out-of-domain data.

In addition to this, we investigated unbalanced mixed-domain training data, and in particular large out-of-domain corpora combined with smaller in-domain corpora. Such a scenario might arise when we have been able to create a small in-domain treebank, but it was not clear in advance how to best make use of that data. We first investigated the naive strategy of concatenating the in-domain and out-of-domain data, which performed solidly and supported the conclusion that in-domain training is far more valuable, although it was never the case that adding large amounts of out-of-domain data was detrimental, even if the benefits were minimal once there was sufficient in-domain data. We also compared two more sophisticated strategies for combining the training data from different domains: linear interpolation between the in-domain and out-of-domain models (Combin), and duplicating the in-domain data some integral number of times (Duplic). The latter was able to significantly improve performance over the naive strategy in the best case, however these methods are parameterised, and a method to select the optimal parameters in advance was required. The technique we presented to achieve this, based on cross-validation over the training data, was reasonably accurate at predicting the optimal training parameters, with a 0.5–1% increase in EDM F-score and 0.5–2% increase in Acc_1 and Acc_{10} in most cases over the naive baseline.

Other work on domain adaptation with limited annotation resources has found that self-training was an effective strategy, so it made sense to evaluate its effectiveness here compared to using a small in-domain treebank. It worked well for improving EDM F-score, with a 1.4–1.5% increase, but often caused a small increase or a large drop in Acc_1 and Acc_{10}. Combining self-training with the best-performing Duplic domain-combination method from above showed similar results, with a 1–1.5% further increase over using Duplic alone for EDM, but no gains or modest decreases for Acc.

We also found some interesting secondary results. As a precursor to evaluating the cross-domain performance penalty, we calculated various inter-corpus and intra-corpus statistics, measuring the similarity of the vocabulary of two corpora using relative entropy and finding vocabulary items which characterise the differences; we also extended these to apply to grammatical constructions rather than lexical items. When the test set and training set were derived from the same domain, they showed a much higher level of similarity than when they were from different domains (as we would expect), and the differences were clearer over words than grammatical constructions.

Overall, we were able to explicitly quantify the domain adaptation penalty with the ERG, as well as providing some estimates of parsing accuracies we can expect over in-domain training data, or in-domain data combined with out-of-domain data. We were also able to provide suggestions for grammar users wishing to adapt to a novel domain to improve parsing accuracy while minimising the amount of treebanking required, by making the best use of in-domain data, and by using a self-training strategy.

8.2.1 Discussion and Future Work

The work in Chapter 4 was important for quantifying the size of the cross-domain performance penalty and in evaluating methods for avoiding it, however there are inevitably areas which could be improved or which could have been done differently in the first place.

We made a decision early on to build the WeScience and LOGON test corpora from sentences randomly picked from the complete corpora, rather than using conventional section boundaries. This was made in order to maximise the difference between in-domain and out-of-domain data for seeing the effects of the cross-domain performance penalty (which was initially the sole focus of the chapter) by ensuring as close as possible a match between in-domain test corpora and training corpora. This almost certainly achieved that effect; the difference between in-domain and out-of-domain would likely have been smaller had the in-domain data been a less close match, which we would expect when using section boundaries rather than shuffled data. We could also justify it on the basis that rather than having the difference between training and test corpora determined on the basis of section boundaries decided somewhat arbitrarily by the corpus creators, the difference between the training and test corpora is controlled by us, and minimised.

However the research later evolved to become concerned additionally with the engineering question of how to best reduce the performance penalty using limited in-domain data. We showed this could be done, and that we could get significant improvements by using a smarter strategy for combining data than the obvious naive strategy, as well as self-training. Given this, the choice of random selection of test data was unfortunate, since the match between training domain and test domain

could be considered artificially close compared to genuinely new test data we could see in the real world. Dividing on a section boundary is arbitrary but may be a more realistic reflection of the differences we would see in new real-world data. Since the improvement in parsing score depends on a close match between training and test data, the effectiveness of methods such as DUPLIC and self-training may have been slightly inflated against what we would see in practice. We kept the same split for comparison with the earlier part of the work, but ideally we would revisit the work using a more realistic training/test split to see if the conclusions still hold.

This problem of needing an artificially close match between training and test data could also be lessened if we had some external technique of evaluating new test data for closeness to an existing domain. The domain profiling research from Section 4.2.2, where we assigned scores to corpora on the basis of their similarity to each other derived from relative entropy, goes some way towards this. If we could apply similar techniques to unannotated test data, we could attempt to match it as closely as possible to a particular training corpus (or some subset of it) before we even get to stage of building a parse selection model. However, our methods assumed the test corpus was already annotated with gold-standard parse trees. For word-based comparison, we could easily use the raw sentence tokens (possibly POS-tagged) instead of the lexical entries from the parse of the sentence, and it is likely the result would be fairly similar. For comparing syntactic constructions, we would need to parse sentences using some existing parse selection model (attempting to find one not strongly biased towards any of the domains we are trying to compare). We would hope that by using the best parse, or maybe several of the highest-ranked parses, we would on average get a good enough reflection of the constructions in use in the domain to compare them to those in another domain. Given that we can get more than 80% EDM F-score without any domain-adaptation work at all, it seems likely that we are getting a large number of constructions correct even in the worst case, so such an approach should be feasible.

We have also not shown that these relative entropy comparison techniques are accurately predictive of parse selection accuracy (although there is almost certainly some correlation). We would need to establish this correlation to know we can usefully select optimal training corpora. Taking this further, we could experiment with these and other corpus comparison scores to determine the optimal combinations of many different corpora, following the approach of McClosky *et al.* (2010) but applied to the ERG rather than the Charniak and Johnson (2005) parser. This would make the question of artificially close domain matches less significant, and greatly increase the ease with which the ERG could be adapted to new domains.

In addition, the work in Chapter 4 showed only indicative results for two domains which happen to be available and provide large quantities of training data. Finding out how broadly applicable they are and whether they extend to other domains leaves room for further research. Building a new custom treebank of perhaps 2000 sentences would be tractable and help answer that question. Another interesting question

is whether we can improve EDM scores, particularly recall, by training a model using semantic dependencies rather than syntactic constituents, and whether these improved scores would be reflected in downstream applications.

Further to this, we have not yet addressed the question of when in the treebanking process it is optimal to build the parse selection model for minimal overall time. A better parse selection model could possibly reduce treebanking time by requiring fewer trees to be examined. For example, from Figure 4.4(a), it is apparent that a domain-adapted model can give as many correct trees in the top 100 as a non-adapted model gives in the top 500. It could also increase the chance of having the correct tree in the parse forest, and this would reduce the need for rejecting trees, which is particularly expensive in the Redwoods treebanking process as it often requires multiple passes through the data (Tanaka *et al.* 2005). It is not clear how important this effect would be, but given this information as well as the time taken to build a model (which is on the order of a few CPU-hours, depending on training set size), we could work out the optimal point in the treebanking process to stop and train a new model to use for the remaining sentences in the corpus. In future work, we are interested in determining where this optimal point lies, based on exploration of the impact of parse selection on determinant selection, and in situ treebanking experiments.

8.3 Treeblazing

In Chapter 5, we considered the problem of creating a domain-adapted treebank for the ERG when a treebank already exists for the target domain, but in an incompatible formalism. Specifically, for the biomedical domain, there are several PTB-style treebanks, including the GENIA Treebank, which was our treebank of choice here. We used this external treebank to assist in the creation of an domain-customised ERG treebank in two ways: to automatically create a treebank in the biomedical domain, or to speed up the treebanking process for humans by reducing the number of constraints they must specify manually. In each case the procedure, denoted *treeblazing*, was similar — we extracted the constituency structure (and, in some cases, node labels) of the GTB for a given sentence then applied a particular set of customised transformations. We compared these extracted constituents with discriminants from the parse forest created for the sentence using the ERG and ruled out conflicting discriminants and hence the corresponding conflicting candidate parse trees. In order to evaluate the parse selection models, we also manually built a small 700-sentence biomedical treebank for the ERG to act as a test corpus.

For automatically building parse selection models, the parse forest was insufficiently reduced by treeblazing alone, so we combined it with a self-training-style strategy to produce training data, and augmented it with manually-annotated WeScience data. The best-performing parse selection model achieved a 1.8% increase in EDM F-score over using WeScience data alone and a 1.2% increase compared to a

standard self-trained model, which in both cases was strongly statistically significant — we were able to obtain a boost in F-score which is probably useful to downstream grammar consumers without any human treebanking labour. On the other hand, the improvements in exact match accuracy were smaller and not significant so the advantages for creating a parse selection model for treebanking are less clear.

However, for using treeblazing to directly speed up treebanking, we also found some encouraging results. The configuration was slightly different, as we did not need to maximally reduce the parse forest — it just needed to be reduced enough to require fewer decisions by the treebankers, and reducing it too much increases the chances of the treeblazing applying an invalid constraint. We aimed to supply the treebankers with a parse forest where 40 trees remained, and we achieved this by using different combinations of the various blazing configurations we had available (including different transformations to the GTB trees, and optionally adding in restrictions derived from labels of leaf nodes or internal nodes). We evaluated the treebanking over a set of 160 sentences supplied to two treebankers, with each sentence either blazed for both annotators, plain for both annotators, or blazed for one annotator only ("half-blazed"). Using the statistics from this, we found that blazing reduced the number of decisions required by 27–43%, the mean annotation time by 8–25% and the median annotation time by 21–26%. In all cases the reductions were more substantial for the less-experienced annotator, who required more decisions and more annotation time in the base case. We also found no evidence of the treeblazing introducing a bias, as the agreement between the annotators on the plain sentences was very close to the agreement for the half-blazed sentences. This indicates that where annotated corpora are available, treeblazing is a viable strategy to reduce treebanking labour (making a fairly inexperienced annotator almost as fast at treebanking as a very experienced annotator).

8.3.1 Discussion and Future Work

While we did discover some very useful results for those wishing to adapt grammars such as the ERG to new domains, it could still be considered that the chapter as a whole was too closely focussed on the GTB, particularly as the annotation guidelines for the test corpus used the GTB annotations as a fallback. This would be easy to fix, with the widespread availability of other biomedical corpora such as PennBioIE (Kulick *et al.* 2004) and the forthcoming CRAFT corpus[1] (Verspoor *et al.* 2012). Either of these could form the basis of a matching ERG treebank to use as a test corpus, or as the source of treeblazing constraints, and this would let us investigate whether the results are applicable across a broader class of biomedical text. In particular, the CRAFT corpus which includes full-text articles could be interesting, as we could determine whether the statistics from abstracts generalise to full-text articles and vice

[1] http://bionlp-corpora.sourceforge.net/CRAFT/

versa, which is important as the textual characteristics of the two are measurably different (Cohen *et al.* 2010), and tools trained on abstracts perform poorly on full-text articles (Verspoor *et al.* 2012). The increasing availability of full-text content from open-access journals means that full-text articles are becoming more viable as an object of study — for example, the BioNLP 2011 shared task (Kim *et al.* 2011) included a component evaluated on full-text. This means that performing well on full-text data is increasingly important.

As well as broadening this work to other aspects of biomedical text, the work in the chapter would in principle be even more generally applicable. There are relatively few customisations specific to the biomedical domain, apart from the transformation we used occasionally to heuristically apply left-bracketing to noun compounds. The procedures we outlined could be applied with little modification wherever there is a PTB-style treebank. One obvious candidate is of course the PTB itself. There is an ongoing project (Kordoni and Zhang 2010) to manually treebank a subset of the PTB WSJ corpus for the ERG, which could potentially benefit from such a speedup to treebanking. It would of course be desirable to derive a gold-standard parsed version of the PTB (as well as other corpora such as the GTB) in a completely automated way, as an analog to CCGBank (Hockenmaier and Steedman 2002). However, our experiments in Chapter 5 indicated that obtaining complete disambiguation using only the annotations of the source corpus would be impossible in most cases, so to create a corpus with a single gold tree, the extra information would need to come from somewhere. It is possible that we could improve, augment or replace the blazing procedure to obtain better disambiguation (although it is not entirely clear how this could be achieved). Failing this, if we get this extra disambiguation from a parse selection model, we will be compromising the quality of the corpus, so having human annotators perform additional disambiguation seems like the only option. In other words, semi-automated transfer of existing corpora may be the only realistic option for the ERG.

Returning to the discussion of the capabilities of treeblazing in the form we have presented it, it may even be possible to apply techniques modelled on treeblazing to other languages or grammatical formalisms. For other languages which have DELPH-IN style HPSG grammars as well as external constituent treebanks, it would be fairly straightforward to adapt the techniques, although the details of the transformations applied to the trees of the external treebank, which are specific to the language and grammar, would need to be revised. For other grammar formalisms, such as CCG, we would need to devise a way to extract discriminants, but the general principles would be equally applicable, with similar requirements to revise ERG-specific transformations of the external trees. Whether such an approach would have an advantage over alternative approaches, some of which we have already outlined, would be an empirical question.

Blazing could also be useful in speeding up the updating process. Redwoods-style treebanks are designed to be dynamic, in that while they are linked to a particular

grammar version, there is an update process to reapply the annotator's decisions with respect to a new grammar and parse forest (Oepen *et al.* 2004). This means that most trees are automatically updated to the new grammar without human intervention. However, there is inevitably a small percentage of trees which need additional annotation, occasionally starting from scratch. By using treeblazing techniques to provide extra disambiguation for treebanks which correspond to external treebanks, it is possible that we could avoid some of the need for extra decisions, and thus speed up the updating of these dynamic treebanks. It is also possible we could refine the treeblazing process itself, for example using a techniques based on the Expectation-Maximisation training procedure of Riezler *et al.* (2002), which was described in Section 2.7.

8.4 Biomedical Information Extraction

Chapter 6 was the first of the application-focussed chapters. We presented a system to apply to task 3 of the BioNLP 2009 shared task, which involved detecting which biomedical events were subject to modification — specifically, speculation and negation. We applied a hybrid approach, involving deep parsing and machine learning, using the ERG and, in some cases, RASP to parse the relevant sentences, then creating rich feature vectors from the RMRS semantics of the sentences as produced by the grammar. These features were generally defined relative to the trigger words of the events. They were a combination of some motivated by intuition and examination of the data (for example outscoping by semantically negative predicates for negation, and outscoping by modals for speculation) and general purpose features. The baseline features were bag-of-words features from a sliding context window, and these features were also used to augment the feature vector in most non-baseline configurations. The feature values were used as input to a maximum entropy learner for classifying events from the training and development sets.

The initial results were encouraging, with in most cases a 1–6% improvement in F-score over the baseline on the development set, depending on the Task 1 input, although in one case the baseline was 3% higher. Over the test set, the feature combination selected on the basis of performance over the development set outperformed the baseline by 3–4%.

In a subsequent round of experimentation, we updated to a newer version of the ERG, and also searched for a correlation between parsing F-score (according to the test corpus from Chapter 5) and performance in this downstream task. Compared to the old parsing model and grammar version, the best performing parse selection models obtained a 3–6% increase in F-score, however the best parse selection models were different depending on the Task 1 input and the kind of modification, and we found no significant correlation between parsing F-score and Task 3 F-score.

8.4.1 Discussion and Future Work

The results of the chapter were reasonably positive. While over the development set, there was one combination of Task 1 inputs and modification type for which we were unable to outperform the bag-of-words baseline, in most cases the features derived from the deep parsing output were found to be useful in detecting modification of events, providing higher F-scores than the naive baseline alone. Indeed, over the test set, the parsing-based approaches were more reliable than the bag-of-words-based approaches, which performed poorly on their own, and had a detrimental effect on performance when added to parsing features. We thus have some (albeit limited) evidence that the parsing-derived features are more generalisable.

The techniques we developed here could be applied to related tasks.For the BioNLP 2011 Shared Task (Kim *et al.* 2011), the system could be run with little modification. However, we would note the caveats from Verspoor *et al.* (2012) about generalising from abstracts to full-text; possibly in this case a new parse selection model adapted for full-text could indeed be useful. The other interesting related task would be the CoNLL 2010 Shared Task (Farkas *et al.* 2010) on detecting scope of modification. It it not clear whether or not a machine learning approach would be the best fit, but as many of our features are targeted at detecting whether some instance of modification includes a particular event trigger, it is likely that many of the feature extraction techniques we used to extract this information from RMRSs could be repurposed for this new task. Additionally we would like to evaluate whether features based on the deep parser outputs could be used for Task 1, which we have not yet attempted using these tools. Since other work, including the work described in Section 2.8.2, has successfully used syntactic features, it would be interesting to evaluate whether using the semantic output from a precision grammar confers any advantages.

8.5 Robust DMRS Inference Rules

In Chapter 7, another application-focussed chapter, we utilised DMRS, a more dependency-like MRS variant. We presented essentially a preemptive solution to the problem of the semantic outputs of a deep parser encoding semantically similar relationships between predicates differently. In a hypothetical DMRS-based information retrieval or information extraction system, recall could be damaged because semantically similar phrases can be represented by different DMRSs, and some of these differences would not be a problem for a bag-of-words-based system. The approach we presented, which was aimed at primarily at noun phrases, was based on anchor text — if a pair of noun endpoints frequently occurs joined by two different paths in many DMRS graphs, there is a good chance the paths are related. By mining a corpus of DMRSs for endpoints and their connecting paths, we built up counts which we could use to learn associations between distinct DMRS subgraphs.

These associations could be used to boost DMRS matching scores between distinct DMRSs in an information retrieval system, but we did not have such a system available, so we instead evaluated by applying the mappings to a new corpus of DMRSs and attempting generation from the created DMRSs. The techniques were somewhat successful — there were various configurations for learning the rules and ranking the most likely mapping candidates based on counts of endpoints (after which a threshold was applied). In the best-performing configurations 10–14% of the rules were able to successfully generate, giving 60–110 mappings in which we could be reasonably confident. At least some of the generated sentences are plausible as paraphrases of the original, lending further support to the validity of the mappings and our techniques for creating them.

8.5.1 Discussion and Future Work

We have already mentioned that to meaningfully evaluate the learnt mappings we need to use the association scores for the mappings in a system which retrieves similar DMRSs or DMRSs with overlapping subgraphs — the generation-based evaluation was an ad hoc stand-in to provide evidence for the plausibility of the approach. One obvious place to apply this would be to SearchBench (Schäfer *et al.* 2011), which already offers DMRS-based querying, although any other corpus of DMRSs could conceivably be used for this. However to see whether there was an advantage to the broadening of the search match provided by the DMRS mappings, we would need a test set of some kind, with manually-annotated relevance judgments for given queries. Evaluating recall in large scale IR systems is notoriously difficult, but we could perform relevance judgements on the extra documents returned by these techniques to evaluate whether the precision was substantially reduced, and estimate the gain in recall against the narrower DMRS-retrieval approach. All of this of course requires annotation time which is not currently available. There also other potential applications, such as Machine Translation systems based on MRSs and semantic transfer, for example Flickinger *et al.* (2005b). Bond *et al.* (2008) noted that paraphrases could be used to broaden MRS matching to trigger semantic MRS-based transfer rules where the match would have otherwise failed; we could possibly achieve a similar effect here using the DMRS mapping rules to broaden the matching criteria. Another possible application might be to Recognising Textual Entailment (Dagan and Glickman 2004). As noted in Section 7.2, Dinu and Wang (2009) found some plausibility for applying inference rules derived from DIRT (Lin and Pantel 2001) to RTE; it is possible something similar could be achieved with the inference rules we have presented here.

There is also likely much room for refinement and expansion of the techniques discussed (which would be more meaningful if the aforementioned task-based evaluation was available). For example, the constraints on the mappings could be relaxed. We could extend them beyond noun compounds — such as expanding to include nominalisations as well, enabling relating the nominalised versions to the verbal version

(so that *synthesis of aspirin* could be related to *Hoffman synthesised aspirin*). We could also expand the class of predicates allowed to link the endpoints, possibly to include open-class words to more closely follow Lin and Pantel (2001)'s original work. Broadening still further, we could attempt to avoid placing any restrictions at all on the nature of the endpoints and the paths joining them, instead building up counts based on statistically-grounded alignments between DMRSs. This may also make the work more cross-linguistically applicable, as there would be fewer language-specific assumptions, although this would need to be tested empirically, and the handling of certain phenomena such as quantifiers would become complicated if we were concerned with producing well-formed DMRSs. A further extension would be to use a less ad hoc method for identifying related paths, instead finding the relationships on the basis of some probabilistic model. We could, for example, postulate a hidden underlying semantics which is realised as several different "surface" semantic forms in various DMRSs, just as there are hidden POS-tags in a Hidden Markov Model POS tagger such as TnT (Brants 2000) discussed in Section 3.2.

Regardless of the rule inference method we use, in these experiments, we also saw interesting effects from using automatically-parsed data for learning the rules instead of sentences treebanked by human. It would be interesting to evaluate more thoroughly how much of an impact the parse quality has on this procedure, such as whether using a better parse selection model could improve the quality of the rules learnt.

8.6 Reflections on Deep Parsing

This thesis has been an exploration of deep parsing with a focus in particular on HPSG and primarily the HPSG-based grammar of English, the ERG. Since the mainstream in parsing in the NLP community uses shallower approaches based on PCFGs, out of necessity we compared and contrasted the commonalities of the approaches, and the differences between them. Each has particular strengths and weaknesses.

In common with other deep parsing formalisms, the ERG has a strong basis in a particular well-established linguistic theory, and it is also relatively computationally intensive to parse (particularly compared to shallow dependency-based approaches). On the other hand, a richer level of annotation output is produced, and the grammar can be more easily updated to reflect a new analysis for some phenomenon based on changes to linguistic theory, rather than being fossilised at the time of treebank creation.

In comparison to the output of treebank parsers based on PCFGs or extensions to them, the output of HPSG shows a number of interesting characteristics. The parse trees are deep in the literal sense, with one or two children per node, compared to the shallower multiple-branching nodes of the PTB. The derivation trees are also deep in the sense of information density, with rich feature structures containing types

drawn from an inheritance hierarchy, embodying a large amount of information. The shapes of these structures are not arbitrary — they reflect the tenets of the underlying HPSG theory which provides the theoretical basis of the grammar. Such structures were postulated as the most satisfactory explanatory mechanism (according to the creators of the theory) for explaining the sentence structures of natural language.

The ERG thus provides a testing ground for a linguistic theory, and this rich structure is undoubtedly interesting from a linguistic perspective. Every correct parse shows that HPSG principles can be applied successfully on a large scale. It is only feasible to build a large corpus of HPSG analyses using a computational grammar such as the ERG, and this large corpus also has potential value to linguists, such as for gathering statistics on usage frequency of particular language constructions. The HPSG analyses here probably have more potential value than a shallower PCFG-based analysis due to their relative linguistic richness and stronger grounding in one particular theoretical framework. In addition, the contents of the grammar itself, through the type hierarchy, lexicon and syntactic rules, provide a valuable resource — it is possible, for example, to determine statistics on noun countability in English using the ERG.

However, in this thesis, we have not primarily concerned ourselves with the linguistic uses of the outputs of the grammar. Such uses are notoriously difficult to evaluate empirically in any case — user studies are complex and expensive at the best of times but when the target audience is a small group such as theoretical linguists, that would add an extra layer of difficulty. But there is also not so much need to justify the ERG in terms of its linguistic value. Rather, we have focussed more on engineering aspects of it, which can be more easily verified in various ways. Part II investigated engineering aspects of parsing sentences themselves, looking at a range of techniques for improving parsing accuracy. More accurate parses can only be regarded as a good thing, whether we intend to use the parses produced for theoretical linguistic analysis, or as the input to some downstream processing component for an external task. In addition, verifying whether there has been an improvement in parsing accuracy is relatively straightforward.

Part III was concerned more with the outputs produced by the parser and how they can be used as a component of a system for some other task — how they are useful for engineering rather than as an object of purely linguistic study. It is usages such as these which are often regarded skeptically by the NLP community; it also (perhaps not coincidentally) not particularly common to use the ERG in such a way. The prevailing attitude is that the shallower approaches do a good enough job at parsing, and that the relatively computationally-intensive parsing process (and possibly also a perceived lack of coverage) does not justify the overhead of using a deep grammar such as the ERG.

For some researchers, including some who reviewed work in this thesis, it is not sufficient to show that deep grammar was useful for a task — we also need to show that it would perform better than a shallower approach. This is not an easy task;

we can sometimes present a plausibility argument showing information which comes from a deep analysis which would not be present in shallower approaches, but to be thoroughly convincing we would need to reimplement a system based on a shallower parser, and even then it would be difficult to be (and be seen to be) unbiased. It also does not seem particularly fair to require evidence that deep parsing is the only way to perform a task — it seems unlikely that someone using the outputs of a shallow constituent parser would be required to justify the extra overhead against using a faster dependency parser. While deep parsing using the ERG may be somewhat time-consuming for parsing terabytes of data (although even that is achievable) the widespread availability of multicore machines and clusters means that parsing times are far less of a concern, particularly when the datasets are only thousands of sentences, as was the case for the IE task in Chapter 6. The maxim "do the simplest thing that could possibly work" is often cited in software engineering; while this may (or may not) be appropriate for producing industrial software, if too many researchers in computer science followed this to its logical end, it could be detrimental to scientific process. Sometimes it is necessary to take risks to see if a more complex approach is useful. If no-one did this, we would probably not bother parsing natural language at all, instead using bag-of-words approaches which get near enough a lot of the time. Even if we do not use all of the extra information produced by the deep parser (which is probably true in many cases), it can still be interesting to see how much of that extra information we can use. Deep parsing may not be the best tool for every task, but it is worth considering in many cases.

8.7 Concluding Remarks

In this thesis we have presented a thorough investigation of deep parsing with the English Resource Grammar, with findings which are potentially useful to grammar developers, treebankers and grammar consumers. With respect to parse selection, we have quantified the cross-domain performance penalty for parse selection, as well as finding ways to avoid the penalty as much as possible while minimising treebanking effort through maximising the value of existing resources.

The ERG has also reached a stage of development where it can be robustly applied to new domains and obtain respectable coverage and parse selection accuracy. We have shown that the information contained in the rich analyses of sentences it produces can be valuable for certain tasks. However, we also hope that quantifying the cross-domain penalty for parse selection and showing ways to avoid it has broadened the range of tasks to which the grammar can be applied, and provided the ability to obtain more accurate analyses in existing systems.

It was not our intention to show that deep parsing tools are the best solution to all problems in natural language processing, or even all problems where syntactic analyses are potentially useful. Indeed, we have not explicitly compared deep and shallow

syntactic approaches on the same task to quantify how useful the extra information is. Such a comparison would be difficult to perform fairly in any case. However, we have hopefully shown that deep parsing can be considered a valuable item in the toolbox of NLP researchers, and contributed in some way to its usefulness.

Bibliography

Abney, Steven. 1996. Statistical methods and linguistics. In *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, ed. by Judith Klavans and Philip Resnik, Cambridge, MA. The MIT Press.

Adolphs, Peter, Stephan Oepen, Ulrich Callmeier, Berthold Crysmann, Dan Flickinger, and Bernd Kiefer. 2008. Some fine points of hybrid natural language parsing. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*, ed. by European Language Resources Association (ELRA), 1380–1387, Marrakech, Morocco.

Airola, Antti, Sampo Pyysalo, Jari Björne, Tapio Pahikkala, Filip Ginter, and Tapio Salakoski. 2008. A graph kernel for protein-protein interaction extraction. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, 1–9, Columbus, Ohio, USA.

Arighi, Cecilia, Zhiyong Lu, Martin Krallinger, Kevin Cohen, W Wilbur, Alfonso Valencia, Lynette Hirschman, and Cathy Wu. 2011. Overview of the BioCreative III workshop. *BMC Bioinformatics* 12(Suppl 8).S1.

Baldwin, Tim, John Beavers, Emily M. Bender, Dan Flickinger, Ara Kim, and Stephan Oepen. 2005. Beauty and the Beast: What running a broad-coverage precision grammar over the BNC taught us about the grammar—and the corpus. *Linguistic evidence: empirical, theoretical, and computational perspectives* 49–70.

Baldwin, Timothy, Colin Bannard, Takaaki Tanaka, and Dominic Widdows. 2003. An empirical model of multiword expression decomposability. In *Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment*, 89–96, Sapporo, Japan.

——, Valia Kordoni, and Aline Villavicencio. 2009. Prepositions in applications: A survey and introduction to the special issue. *Computational Linguistics* 35(2).119–149.

Bangalore, Srinivas, and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics* 25(2).

Barzilay, Regina, and Lillian Lee. 2003. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proceedings of the 2003 Human Language Technology Conference of the North American Association for Computational Lingustics (HLT-NAACL 2003)*, Berkeley, Calfornia, USA.

Benson, Steven J., and Jorge J. More. 2001. A limited memory variable metric method in subspaces and bound constrained optimization problems. Technical report, Argonne National Laboratory.

Berger, Adam L., Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics* 22(1).

Bies, Ann, Mark Ferguson, Karen Katz, Robert MacIntyre, Victoria Tredinnick, Grace Kim, Mary Ann Marcinkiewicz, and Britta Schasberger. 1995. Bracketing guidelines for Treebank II style Penn Treebank project. Technical report, University of Pennsylvania.

Bikel, Daniel M. 2002. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proceedings of the second international conference on Human Language Technology Research*, 178–182, San Diego, California.

Björne, Jari, Juho Heimonen, Filip Ginter, Antti Airola, Tapio Pahikkala, and Tapio Salakoski. 2009. Extracting complex biological events with rich graph-based feature sets. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, 10–18, Boulder, Colorado, USA.

——, and Tapio Salakoski. 2011. Generalizing biomedical event extraction. In *Proceedings of BioNLP Shared Task 2011 Workshop*, 183–191, Portland, Oregon, USA.

Black, Ezra, Steven Abney, Dan Flickinger, Claudia Gdaniec, Ralph Grishman, Philip Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith Klavans, Mark Liberman, Mitch Marcus, Salim Roukos, Beatrice Santorini, and Tomek Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the February 1991 DARPA Speech and Natural Language Workshop*, 306–311, Pacific Grove, USA.

Blum, Avrim, and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 92–100.

Bond, Francis, Sanae Fujita, Chikara Hashimoto, Kaname Kasahara, Shigeko Nariyama, Eric Nichols, Akira Ohtani, Takaaki Tanaka, and Shigeaki Amano. 2004. The hinoki treebank: A treebank for text understanding. In *Proceedings of the 1st International Joint Conference on Natural Language Processing (IJCNLP 2004)*, 554–559, Hainan, China.

——, Eric Nichols, Darren Scott Appling, and Michael Paul. 2008. Improving statistical machine translation by paraphrasing the training data. In *Proceedings of the International Workshop on Spoken Language Translation*, 150–157, Hawaii, USA.

Bos, Johan, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International conference on Computational Linguistics (COLING 2004)*, 1240–1246, Geneva, Switzerland.

Brants, Sabine, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. 2004. TIGER: Linguistic interpretation of a German corpus. *Research on Language & Computation* 2.597–620.

Brants, Thorsten. 2000. TnT – a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing (ANLP 2000)*, 224–231, Seattle, Washington, USA.

Bresnan, Joan. 2000. *Lexical-functional Syntax*. Blackwell textbooks in linguistics: 16. Malden, Massachussets, USA: Blackwell.

Briscoe, Ted, John Carroll, and Rebecca Watson. 2006. The second release of the RASP system. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, 77–80, Sydney, Australia.

——, and Ann Copestake. 1999. Lexical rules in constraint-based grammars. *Computational Linguistics* 25(4).

Burnard, Lou. 2000. User reference guide for the british national corpus. Technical report, Oxford University Computing Services.

Butt, Miriam, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. 2002. The parallel grammar project. In *Proceedings of COLING 2002 Workshop on Grammar Engineering and Evaluation*, 1–7, Taipei, Taiwan.

Buyko, Ekaterina, Erik Faessler, Joachim Wermter, and Udo Hahn. 2009. Event extraction from trimmed dependency graphs. In *Proceedings of the*

BioNLP 2009 Workshop Companion Volume for Shared Task, 19–27, Boulder, Colorado, USA.

——, Joachim Wermter, Michael Poprat, and Udo Hahn. 2006. Automatically adapting an NLP core engine to the biology domain. In *Proceedings of the Joint BioLINK-Bio-Ontologies Meeting. A Joint Meeting of the ISMB Special Interest Group on Bio-Ontologies and the BioLINK Special Interest Group on Text Data M ining in Association with ISMB*, 65–68.

Cahill, Aoife, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith, and Andy Way. 2008. Wide-coverage deep statistical parsing using automatic dependency structure annotation. *Computational Linguistics* 34(1).81–124.

——, Mairéad Mccarthy, Josef van Genabith, and Andy Way. 2002. Parsing with PCFGs and automatic F-Structure. In *The Proceedings of the 2002 Conference on LFG*, ed. by Miriam Butt and Tracy Holloway King, 76–95.

Callmeier, Ulrich. 2000. Pet – a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering* 6(1).99–107.

Carroll, J, Ann Copestake, Dan Flickinger, and Victor Poznanski. 1999. An efficient chart generator for (semi-) lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation (EWNLG'99)*, 86–95.

Carroll, John, and Stephan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing*, 165–176, Jeju Island, Korea.

Carter, David. 1997. The TreeBanker. A tool for supervised training of parsed corpora. In *Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, 9–15.

Chapman, Wendy W., Will Bridewell, Paul Hanbury, Gregory F. Cooper, and Bruce G. Buchanan. 2001. A simple algorithm for identifying negated findings and diseases in discharge summaries. *Journal of Biomedical Informatics* 34(5).301–310.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Conference of the North American chapter of the Association for Computational Linguistics (NAACL 2000)*, 132–139, Seattle, Washington, USA.

Charniak, Eugene. 1996. Tree-bank grammars. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 1996)*, 1031–1036, Portland, Oregon, USA.

——. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI 1997)*, 598–603, Providence, Rhode Island, USA.

——, and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, 173–180, Ann Arbor, Michigan, USA.

Chen, Stanley, and Ronald Rosenfeld. 1999. A gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Carnegie Mellon University.

Chomsky, Noam. 1957. *Syntactic structures*. Janua linguarum: Series minor ur 4. The Hague, Netherlands: Mouton.

——. 1965. *Aspects of the Theory of Syntax*. Cambridge, Massachusetts, USA: The MIT Press.

——. 1982. *Some concepts and consequences of the theory of government and binding*. Linguistic inquiry monographs: 6. Cambridge, Massachusetts, USA: MIT Press.

Clark, Alexander. 2001. Unsupervised induction of stochastic context-free grammars using distributional clustering. In *Proceedings of the Fifth Conference on Computational Natural Language Learning (CoNLL 2001)*, 105–112.

Clark, Stephen, and James Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, 104–111.

——, and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics* 33(4).493–552.

——, and Julia Hockenmaier. 2002. Evaluating a wide-coverage CCG parser. In *Proceedings of 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, 60–66.

——, ——, and Mark Steedman. 2002. Building deep dependency structures using a wide-coverage CCG parser. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, 327–334, Philadelphia, Pennsylvania, USA.

Clegg, Andrew B., and Adrian J. Shepherd. 2005. Evaluating and integrating treebank parsers on a biomedical corpus. In *Proceedings of the Workshop on Software*, 14–33.

Cohen, K Bretonnel, Helen Johnson, Karin Verspoor, Christophe Roeder, and Lawrence Hunter. 2010. The structural and content aspects of abstracts versus bodies of full text journal articles are different. *BMC Bioinformatics* 11(1).492.

Cohen, Paul R. 1995. *Empirical methods for artificial intelligence*. Cambridge, Massachusetts, USA: MIT Press.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, 16–23, Madrid, Spain.

——, 1999. *Head-Driven Statistical Models for Natural Language Parsing*. University of Pennsylvania dissertation.

Collins, Michael John. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 184–191, Santa Cruz, California, USA.

Copestake, Ann. 2004. Report on the design of RMRS. Technical Report D1.1a, University of Cambridge, Cambridge, UK.

——. 2009. Slacker semantics: Why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2009)*, 1–9, Athens, Greece.

——, and Dan Flickinger. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)*, Athens, Greece.

——, Dan Flickinger, Ivan A. Sag, and Carl Pollard. 2005. Minimal recursion semantics: An introduction. *Research on Language & Computation* 3(4).281–332.

——, Alex Lascarides, and Dan Flickinger. 2001. An algebra for semantic construction in constraint-based grammars. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, 140–147.

Crouch, Dick, Mary Dalrymple, Ron Kaplan, Tracy King, John Maxwell, and Paula Newman. 2008. Xle documentation. Technical report, Palo Alto Research Center.

Crysmann, Berthold. 2005. Relative clause extraposition in German: An efficient and portable implementation. *Research on Language & Computation* 3(1).61–82.

Curran, James R., and Stephen Clark. 2003. Investigating gis and smoothing for maximum entropy taggers. In *Proceedings of the 10th Meeting of the European Chapter of the Association for Computational Linguistics*, 91–98,, Budapest, Hungary.

Dagan, Ido, and Oren Glickman. 2004. Probabilistic textual entailment: Generic applied modeling of language variability. In *PASCAL Workshop on Learning Methods for Text Understanding and Mining*, Grenoble, France.

——, ——, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. *Lecture Notes in Computer Science* 3944.177–190.

De Marneffe, Marie-Catherine, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, 449–454, Genoa, Italy.

Dinu, Georgiana, and Rui Wang. 2009. Inference rules and their application to recognizing textual entailment. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2009)*, 211–219, Athens, Greece.

Dridan, Rebecca, and Timothy Baldwin. 2010. Unsupervised parse selection for HPSG. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 694–704, Cambridge, Massachusetts, USA.

——, and Stephan Oepen. 2011. Parser evaluation using elementary dependency matching. In *Proceedings of the 12th International Conference on Parsing Technologies*, 225–230, Dublin, Ireland.

Farkas, Richárd, Veronika Vincze, György Móra, János Csirik, and György Szarvas. 2010. The CoNLL-2010 shared task: Learning to detect hedges and their scope in natural language text. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL 2010)*, 1–12, Uppsala, Sweden.

Flickinger, Dan. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering* 6 (1).15–28.

——, Richa Bhayani, and Stanley Peters. 2009. Sentence boundary detection in spoken dialogue. Technical report, Stanford University, TR-09-06.CSLI.

——, Alexander Koller, and Stefan Thater. 2005a. A new well-formedness criterion for semantics debugging. In *Proceedings of the 12th International Conference on HPSG*, 129–142, Lisbon, Portugal.

——, Jan Tore Lønning, Helge Dyvik, Stephan Oepen, and Francis Bond. 2005b. SEM-I rational MT — enriching deep grammars with a semantic interface for scalable machine translation. In *Proceedings of MT Summit X*, 165–172, Phuket, Thailand.

——, Stephan Oepen, and Gisle Ytrestøl. 2010. Wikiwoods: Syntacto-semantic annotation for English Wikipedia. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*, Valetta, Malta.

Forst, Martin. 2003. Treebank conversion - establishing a testsuite for a broad-coverage LFG from the TIGER treebank. In *Proceedings of 4th International Workshop on Linguistically Interpreted Corpora (LINC-03) at EACL 2003*, Budapest, Hungary.

Francis, W. Nelson. 1979. A standard sample of present-day English for use with digital computers (revised and amplified from original 1964 edition). Report to the U.S Office of Education on Cooperative Research Project No. E-007, Brown University., Providence, Rhode Island, USA.

Frank, Anette. 2004. Constraint-based RMRS construction from shallow grammars. In *Proceedings of the 20th International conference on Computational Linguistics (COLING 2004)*, 1269–1272, Geneva, Switzerland.

Garside, Roger, Geoffrey Leech, and Anthony McEnery (eds.) 1997. *Corpus Annotation: Linguistic Information from Computer Text Corpora*. New York, New York, USA: Addison Wesley Longman Ltd.

Gazdar, Gerald. 1985. *Generalized phrase structure grammar*. Oxford, United Kingdom: B. Blackwell.

Giampiccolo, Danilo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, 1–9, Prague, Czech Republic.

Gildea, Daniel. 2001. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 167–202, Pittsburgh, USA.

——, and Dan Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics* 28(3).245–288.

Giménez, Jesús, and Lluís Màrquez. 2004. SVMTool: A general POS tagger generator based on support vector machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, 43–46, Lisbon, Portugal.

Hajič, Jan. 2006. Complex corpus annotation: The prague dependency treebank. In *Insight into the Slovak and Czech Corpus Linguistics*, ed. by Mária Simková, chapter 5, p. 54. Bratislava, Slovakia: VEDA.

Hara, Tadayoshi, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Adapting a probabilistic disambiguation model of an HPSG parser to a new domain. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP 2005)*, 99–210, Jeju Island, Korea.

——, ——, and ——. 2007. Evaluating impact of re-training a lexical disambiguation model on domain adaptation of an HPSG parser. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT 2007)*, 11–22, Prague, Czech Republic.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2001. *Elements of Statistical Learning*. New York, New York, USA: Springer-Verlag.

Hirschman, Lynette, Alexander Yeh, Christian Blaschke, and Alfonso Valencia. 2005. Overview of BioCreAtIvE: critical assessment of information extraction for biology. *BMC Bioinformatics* 6(Suppl 1).S1.

Hockenmaier, Julia, and Mark Steedman. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, Las Palmas, Spain.

Honnibal, Matthew, Joel Nothman, and James R. Curran. 2009. Evaluating a statistical CCG parser on Wikipedia. In *People's Web '09: Proceedings of the 2009 Workshop on The People's Web Meets NLP*, 38–41, Suntec, Singapore.

Huddleston, Rodney. 1988. *English Grammar: An Outline*. Cambridge University Press.

Jackson, Peter, and Isabelle Moulinier. 2007. *Natural Language Processing for Online Applications*, volume 5 of *Natural Language Processing*. Amsterdam: John Benjamins Publishing Co., 2nd revised edition.

Joachims, Thorsten, Thomas Finley, and Chu-Nam John Yu. 2009. Cutting-plane training of structural SVMs. *Machine Learning* 77(1).27–59.

Johnson, Mark. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, 136–143, Philadelphia, Pennsylvania, USA.

——, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 535–541, College Park, Maryland, USA.

Joshi, Aravind K., and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, ed. by Grzegorz Rozenberg and Arto Salomaa, volume 3. Berlin, Germany: Springer-Verlag.

Jurafsky, Daniel, and James Martin. 2000. *Speech and Language Processing*. Upper Saddle River, New Jersey, USA: Prentice-Hall.

Kay, Martin. 1986. Algorithm schemata and data structures in syntactic processing. In *Readings in Natural Language Processing*, ed. by Barbara J. Grosz, Karen Sparck Jones, and Bonnie Lynn Webber. Los Altos, Calif., USA: Morgan Kaufmann Publishers Inc.

Kiefer, Bernd, Hans-Ulrich Krieger, John Carroll, and Rob Malouf. 1999. A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 473–480, College Park, Maryland, USA.

Kilgarriff, Adam. 2004. How dominant is the commonest sense of a word? In *Text, Speech and Dialogue*, Lecture Notes in Artificial Intelligence Vol. 3206, 103–112.

Kilicoglu, Halil, and Sabine Bergler. 2008. Recognizing speculative language in biomedical research articles: a linguistically motivated perspective. *BMC Bioinformatics* 9(Suppl 11).S10.

——, and ——. 2009. Syntactic dependency based heuristics for biological event extraction. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, 119–127, Boulder, Colorado, USA.

——, and ——. 2011. Adapting a general semantic interpretation approach to biological event extraction. In *Proceedings of BioNLP Shared Task 2011 Workshop*, 173–182, Portland, Oregon, USA.

Kim, Jin-Dong, Tomoko Ohta, Sampo Pyysalo, Yoshinobu Kano, and Jun'ichi Tsujii. 2009. Overview of bionlp'09 shared task on event extraction. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, 1–9, Boulder, Colorado, USA.

——, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. 2003. GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics* 19(Supplement 1).i180–i182.

——, Tomoko Ohta, Yuka Teteisi, and Jun'ichi Tsujii. 2006. GENIA ontology. Technical Report TR-NLP-UT-2006-2.

——, Tomoko Ohta, and Jun'ichi Tsujii. 2008. Corpus annotation for mining biomedical events from literature. *BMC Bioinformatics* 9.10.

——, Tomoko Ohta, Yoshimasa Tsuruoka, Yuka Tateisi, and Nigel Collier. 2004. Introduction to the bio-entity recognition task at JNLPBA. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP 2004)*, 70–75, Geneva, Switzerland.

——, Yue Wang, Toshihisa Takagi, and Akinori Yonezawa. 2011. Overview of genia event task in bionlp shared task 2011. In *Proceedings of BioNLP Shared Task 2011 Workshop*, 7–15, Portland, Oregon, USA.

King, Tracy Holloway, Richard Crouch, Stefan Riezler, Mark Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 dependency bank. In *Proceedings of 4th International Workshop on Linguistically Interpreted Corpora (LINC-03) at EACL 2003*, 1–8, Budapest, Hungary.

Klein, Dan, and Christopher D. Manning. 2002. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL 2002)*, 128–135, Philadelphia, Pennsylvania.

——, and ——. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, 423–430, Sapporo, Japan.

Koeling, Rob, Diana McCarthy, and John Carroll. 2005. Domain-specific sense distributions and predominant sense acquisition. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 419–426, Vancouver, British Columbia, Canada.

Kordoni, Valia, and Yi Zhang. 2010. Disambiguating compound nouns for a dynamic HPSG treebank of wall street journal texts. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*, Valetta, Malta.

Krallinger, Martin, Alexander Morgan, Larry Smith, Florian Leitner, Lorraine Tanabe, John Wilbur, Lynette Hirschman, and Alfonso Valencia. 2008. Evaluation of text-mining systems for biology: overview of the second BioCreative community challenge. *Genome Biology* 9(Suppl 2).S1.

Krieger, Hans-Ulrich, and Ulrich Schäfer. 1994. TDL – a type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING 1994)*, 893–899, Kyoto, Japan.

Kulick, Seth, Ann Bies, Mark Liberman, Mark Mandel, Ryan McDonald, Martha Palmer, Andrew Schein, Lyle Ungar, Scott Winters, and Pete White. 2004. Integrated annotation for biomedical information extraction. In *HLT-NAACL 2004 Workshop: BioLINK 2004, Linking Biological Literature, Ontologies and Databases*, ed. by Lynette Hirschman and James Pustejovsky, 61–68, Boston, Massachusetts, USA.

Kupiec, Julian. 1992. An algorithm for estimating the parameters of unrestricted hidden stochastic context-free grammars. In *Proceedings of the 14th conference on Computational Linguistics (COLING 1992)*, 387–393.

Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, 282–289, Williamstown, Massachussets, USA.

Lauer, Mark, 1996. *Designing statistical language learners: experiments on noun compounds*.

Lease, Matthew, and Eugene Charniak. 2005. Parsing biomedical literature. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP 2005)*, 58–69, Jeju Island, Korea.

Lin, Dekang, and Patrick Pantel. 2001. DIRT - discovery of inference rules from text. In *In Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 323–328, San Francisco, California, USA.

Lønning, Jan Tore, Stephan Oepen, Dorothee Beermann, Lars Hellan, John Carroll, Helge Dyvik, Dan Flickinger, Janne Bondi Johannessen, Paul Meurer, Torbjørn Nordgård, Victoria Rosén, and Erik Velldal. 2004. LOGON. A Norwegian MT effort. In *Proceedings of the Workshop in Recent Advances in Scandinavian Machine Translation*, Uppsala, Sweden.

MacKinlay, Andrew, 2005. The effects of part-of-speech tagsets on tagger performance. Honours thesis, University of Melbourne.

——, Rebecca Dridan, Dan Flickinger, Stephan Oepen, and Timothy Baldwin. 2011a. Treeblazing: Using external treebanks to filter parse forests for parse selection and treebanking. In *Proceedings of the 5th International Joint*

Conference on Natural Language Processing (IJCNLP 2011), 246–254, Chiang Mai, Thailand.

——, David Martinez, and Timothy Baldwin. 2009. Biomedical event annotation with CRFs and precision grammars. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, 77–85, Boulder, Colorado, USA.

——, ——, and ——. 2011b. A parser-based approach to detecting modification of biomedical events. In *Proceedings of the ACM Fourth International Workshop on Data and Text Mining in Biomedical Informatics (DTMBIO 2011)*, 51–58, Glasgow, United Kingdom.

Magerman, David M. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 276–283, Cambridge, Massachusetts, USA.

Malouf, Robert. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL 2002)*, 49–55, Taipei, Taiwan.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. 2008. *An Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press.

——, and Hinrich Schütze. 2000. *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts, USA: The MIT Press.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the Penn Treebank. *Computational Linguistics* 19(2).313–330.

Masuichi, Hiroshi, Tomoko Ohkuma, Hiroki Yoshimura, and Yasunari Harada. 2003. Japanese parser on the basis of the lexical-functional grammar formalism and its evaluation. *Journal of Natural Language Processing* 10(2).79–109.

McClosky, David, 2010. *Any Domain Parsing: Automatic Domain Adapatation for Parsing*. Brown dissertation.

——, and Eugene Charniak. 2008. Self-training for biomedical parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, 101–104, Sydney, Australia.

——, ——, and Mark Johnson. 2006a. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, 152–159, New York, New York, USA.

——, ——, and Mark Johnson. 2006b. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, 337–344, Sydney, Australia.

——, ——, and Mark Johnson. 2010. Automatic domain adaptation for parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 28–36, Los Angeles, California, USA.

——, Mihai Surdeanu, and Christopher Manning. 2011. Event extraction as dependency parsing for BioNLP 2011. In *Proceedings of BioNLP Shared Task 2011 Workshop*, 41–45, Portland, Oregon, USA.

McCray, Alexa T., Surash Srinivasan, and Allen C. Browne. 1994. Lexical methods for managing variation in biomedical terminologies. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, 235–239, Washington, DC, USA.

McDonald, Ryan, and Fernando Pereira. 2005. Identifying gene and protein mentions in text using conditional random fields. *BMC Bioinformatics* 6(Suppl 1).S6.

Miller, George A. 1995. WordNet: a lexical database for English. *Communications of the ACM* 38(11).39–41.

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the Penn Treebank. In *Proceedings of the 1st International Joint Conference on Natural Language Processing (IJCNLP 2004)*, 684–693, Hainan, China.

——, Kenji Sagae, Rune Saetre, Takuya Matsuzaki, and Jun'ichi Tsujii. 2009. Evaluating contributions of natural language parsers to protein-protein interaction extraction. *Bioinformatics* 25(3).394–400.

——, and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, 83–90, Ann Arbor, Michigan.

——, and ——. 2008. Feature forest models for probabilistic HPSG parsing. *Computational Linguistics* 34(1).35–80.

Müller, Stephan, and Walter Kasper. 2000. HPSG analysis of german. In *Verbmobil : foundations of speech-to-speech translation*, ed. by Wolfgang Wahlster. Berlin, Germany: Springer-Verlag.

Nakov, Preslav, and Marti Hearst. 2005. Search engine statistics beyond the n-gram: application to noun compound bracketing. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL 2005)*, 17–24, Ann Arbor, Michigan.

——, and Hwee Tou Ng. 2009. Improved statistical machine translation for resource-poor languages using related resource-rich languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 1358–1367, Singapore.

Ng, Andrew, and Michael I. Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Proceedings of the 14th Conference on Advances in Neural Information Processing Systems 14 (NIPS 2002)*, volume 2, 841–848, Vancouver, British Columbia, Canada.

Nigam, Kamal, and Rayid Ghani. 2000. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM 2009)*, 86–93, Hong Kong, China.

Niu, Zheng-Yu, Haifeng Wang, and Hua Wu. 2009. Exploiting heterogeneous treebanks for parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2009)*, 46–54, Suntec, Singapore.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, 2216–2219, Genoa, Italy.

Oepen, Stephan. 2001. [incr tsdb()] — Competence and performance laboratory. User manual. Technical report, Saarland University, Saarbrücken, Germany.

——, and John Carroll. 2000. Parser engineering and performance profiling. *Natural Language Engineering* 6(01).81–97.

——, Dan Flickinger, Kristina Toutanova, and Christopher D. Manning. 2004. LinGO Redwoods: A rich and dynamic treebank for HPSG. *Research on Language & Computation* 2(4).575–596.

——, and Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, Genoa, Italy.

——, Kristina Toutanova, Stuart M. Shieber, Christopher D. Manning, Dan Flickinger, and Thorsten Brants. 2002. The LinGO Redwoods treebank: Motivation and preliminary applications. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING 2002)*, 1–5, Taipei, Taiwan.

O'Grady, William, Michael Dobrovolsky, and Francis Katamba. 1997. *Contemporary Linguistics: An Introduction*. Pearson Education Ltd.

Pang, Bo, Kevin Knight, and Daniel Marcu. 2003. Syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, 102–109, Berkeley, Calfornia, USA.

Plank, Barbara, and Gertjan van Noord. 2008. Exploring an auxiliary distribution based approach to domain adaptation of a syntactic disambiguation model. In *Proceedings of the COLING 2008 Workhop on Cross Framework and Cross Domain Parser Evaluation*, 9–16, Manchester, UK.

Pollard, Carl, and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. University of Chicago Press.

Pullum, Geoffrey K., and Gerald Gazdar. 1982. Natural languages and context-free languages. *Linguistics and Philosophy* 4(4).471–504.

Pyysalo, Sampo, Filip Ginter, Juho Heimonen, Jari Bjorne, Jorma Boberg, Jouni Jarvinen, and Tapio Salakoski. 2007. BioInfer: A corpus for information extraction in the biomedical domain. *BMC Bioinformatics* 8(1).50.

Ramshaw, Lance, and Mitchell P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*, 82–94, Cambridge, Massachusetts, USA.

Ratnaparkhi, Adwait. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the 1996 Conference on Empirical Methods in Natural Language Processing (EMNLP 1996)*, 133–142, Philadelphia, USA.

——, 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. University of Pennsylvania dissertation.

——, Jeffrey C. Reynar, and Salim Roukos. 1994. A maximum entropy model for prepositional phrase attachment. In *Proceedings of the conference on Human Language Technology (HLT 1994)*, 250–255, Plainsboro, New Jersey, USA.

Rayson, Paul, and Roger Garside. 2000. Comparing corpora using frequency profiling. In *The Workshop on Comparing Corpora*, 1–6, Hong Kong, China.

Riedel, Sebastian, Hong-Woo Chun, Toshihisa Takagi, and Jun'ichi Tsujii. 2009. A markov logic approach to bio-molecular event extraction. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, 41–49, Boulder, Colorado, USA.

——, and Andrew McCallum. 2011. Robust biomedical event extraction with dual decomposition and minimal domain adaptation. In *Proceedings of BioNLP Shared Task 2011 Workshop*, 46–50, Portland, Oregon, USA.

Riezler, Stefan, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL 2002)*, 271–278, Philadelphia, Pennsylvania.

Rimell, Laura, and Stephen Clark. 2009. Porting a lexicalized-grammar parser to the biomedical domain. *Journal of Biomedical Informatics* 42(5).852–865.

Roark, Brian, and Michiel Bacchiani. 2003. Supervised and unsupervised PCFG adaptation to novel domains. In *Proceedings of the 2003 Human Language Technology Conference of the North American Association for Computational Lingustics (HLT-NAACL 2003)*, 126–133, Berkeley, Calfornia, USA.

Rosenberg, Chuck, Martial Hebert, and Henry Schneiderman. 2005. Semi-supervised self-training of object detection models. In *Seventh IEEE workshop on applications of computer vision*, volume 1, 29–36.

Rush, Alexander M, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 1–11, Cambridge, Massachusetts, USA.

Sætre, Rune, Kazuhiro Yoshida, Akane Yakushiji, Yusuke Miyao, Yuichiro Matsubayashi, and Tomoko Ohta. 2007. AKANE system: Protein-protein interaction pairs in the BioCreAtIvE2 Challenge, PPI-IPS subtask. In *Proceedings of the Second BioCreative Challenge Evaluation Workshop*, Madrid, Spain.

Sag, Ivan A., Thomas Wasow, and Emily M. Bender. 2003. *Syntactic Theory: A formal introduction*. Leland Stanford Junior University: CSLI Publications.

Sagae, Kenji. 2010. Self-training without reranking for parser domain adaptation and its impact on semantic role labeling. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, 37–44, Uppsala, Sweden.

Salomaa, Arto. 1973. *Formal Languages*. New York; San Francisco; London: Academic Press.

Sasaki, Yutaka, Yoshimasa Tsuruoka, John McNaught, and Sophia Ananiadou. 2008. How to make the most of ne dictionaries in statistical ner. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, 63–70, Columbus, Ohio, USA.

Schäfer, Ulrich, Bernd Kiefer, Christian Spurk, Jörg Steffen, and Rui Wang. 2011. The acl anthology searchbench. In *Proceedings of the ACL-HLT 2011 System Demonstrations*, 7–13, Portland, Oregon, USA.

Schlangen, David, 2003. *A Coherence-Based Approach to the Interpretation of Non-Sentential Utterances in Dialogue*. School of Informatics, University of Edinburgh dissertation.

Sells, Peter. 1985. *Lectures on contemporary syntactic theories : an introduction to government-binding theory, generalized phrase structure grammar, and lexical-functional grammar*. CSLI lecture notes: no. 3. Stanford, California, USA: Center for the Study of Language and Information, Stanford University.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8(3).pp. 333–343.

Siegel, Melanie, and Emily M. Bender. 2002. Efficient deep processing of japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization*, 1–8.

Snow, Rion, Daniel Jurafsky, and Andrew Y. Ng. 2005. Learning syntactic patterns for automatic hypernym discovery. In *Proceedings of the 17th Conference on Advances in Neural Information Processing Systems (NIPS 2005)*, Vancouver, British Columbia, Canada.

Somers, Harold L. 1984. On the validity of the complement-adjunct distinction in valency grammar. *Linguistics* 22(4).507.

Steedman, Mark. 2000. *The syntactic process*. Language, speech, and communication. Cambridge, Massachusetts, USA: MIT Press.

Stevenson, Suzanne, Afsaneh Fazly, and Ryan North. 2004. Statistical measures of the semi-productivity of light verb constructions. In *Second ACL Workshop on Multiword Expressions: Integrating Processing*, ed. by Takaaki Tanaka, Aline Villavicencio, Francis Bond, and Anna Korhonen, 1–8, Barcelona, Spain.

Tanaka, Takaaki, Francis Bond, Stephan Oepen, and Sanae Fujita. 2005. High precision treebanking—blazing useful trees using POS information. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, 330–337, Ann Arbor, Michigan.

Tateisi, Yuka, and Jun'ichi Tsujii. 2004. Part-of-speech annotation of biology research abstracts. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*.

——, Akane Yakushiji, Tomoko Ohta, and Jun'ichi Tsujii. 2005. Syntax annotation for the GENIA corpus. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP 2005)*, 222–227, Jeju Island, Korea.

Toutanova, Kristina, Dan Klein, Christoper D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *North American Chapter of the Association for Computational Linguistics*, 173–180, Edmonton, Canada.

——, Christopher Manning, Dan Flickinger, and Stephan Oepen. 2005. Stochastic HPSG parse disambiguation using the Redwoods Corpus. *Research on Language & Computation* 3(1).83–105.

Tsuruoka, Yoshimasa, Yuka Tateishi, Jin-Dong Kim, Tomoko Ohta, John McNaught, Sophia Ananiadou, and Jun'ichi Tsujii. 2005. Developing a robust part-of-speech tagger for biomedical text. In *Advances in Informatics: 10th Panhellenic Conference on Informatics*, 382–392, Volos, Greece.

——, and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 467–474, Vancouver, British Columbia, Canada.

Vadas, David, and James Curran. 2007. Adding noun phrase structure to the Penn Treebank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 240–247, Prague, Czech Republic.

van der Beek, Leonoor, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002. The Alpino dependency treebank. *Computational Linguistics in the Netherlands* 45(1).8–22.

Van Landeghem, Sofie, Yvan Saeys, Bernard De Baets, and Yves Van de Peer. 2009. Analyzing text in search of bio-molecular events: a high-precision machine learning framework. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, 128–136, Boulder, Colorado, USA.

van Noord, Gertjan, and Robert Malouf. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of Workshop Beyond Shallow Analyses: Formalisms and statistical modeling for deep analyses at IJCNLP 2004*, Hainan, China.

Velldal, Erik, 2007. *Empirical Realization Ranking*. University of Oslo Department of Informatics dissertation.

——, Lilja Øvrelid, and Stephan Oepen. 2010. Resolving speculation: Maxent cue classification and dependency-based scope rules. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL 2010)*, 48–55, Uppsala, Sweden.

Verspoor, Karin, K. Bretonnel Cohen, Ben Goertzel, and Inderjeet Mani. 2006. Linking natural language processing and biology: Towards deeper biological literature analysis (preface). In *BioNLP'06*.

——, K Bretonnel Cohen, and Lawrence Hunter. 2009. The textual characteristics of traditional and open access scientific journals are similar. *BMC Bioinformatics* 10(1).183.

——, K. Bretonnel Cohen, Arrick Lanfranchi, Colin Warner, Helen L. Johnson, Christophe Roeder, Jinho D. Choi, Christopher Funk, Yuriy Malenkiy, Miriam Eckert, Nianwen Xue, William A. Baumgartner Jr., Michael Bada, Martha Palmer, , and Lawrence E. Hunter. 2012. A corpus of full-text journal articles is a robust evaluation tool for revealing differences in performance of biomedical natural language processing tools (in press). *BMC Bioinformatics* .

Vincze, Veronika, György Szarvas, Richárd Farkas, György Móra, and János Csirik. 2008. The BioScope corpus: biomedical texts annotated for uncertainty, negation and their scopes. *BMC Bioinformatics* 9(Suppl 11).S9.

Vlachos, Andreas. 2010. Two strong baselines for the BioNLP 2009 event extraction task. In *Proceedings of the 2010 Workshop on Biomedical Natural Language Processing*, 1–9, Uppsala, Sweden.

——, Paula Buttery, Diarmuid Ó Séaghdha, and Ted Briscoe. 2009. Biomedical event extraction without training data. In *Proceedings of the BioNLP*

2009 Workshop Companion Volume for Shared Task, 37–40, Boulder, Colorado, USA.

Wahlster, Wolfgang (ed.) 2000. *Verbmobil : foundations of speech-to-speech translation*. Berlin, Germany: Springer-Verlag.

Wang, Jong-Nae, Jing-Shin Chang, and Keh-Yih Su. 1994. An automatic treebank conversion algorithm for corpus sharing. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, 248–254, Las Cruces, New Mexico, USA.

White, Michael. 2011. Glue rules for robust chart realization. In *Proceedings of the 13th European Workshop on Natural Language Generation*, 194–199, Nancy, France.

Witten, Ian H., and Eibe Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco, California, USA: Morgan Kaufmann Publishers Inc.

Wubben, Sander, Erwin Marsi, Antal van den Bosch, and Emiel Krahmer. 2011. Comparing phrase-based and syntax-based paraphrase generation. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, 27–33, Portland, Oregon, USA.

Xia, Fei. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS 1999)*, 398–403, Beijing, China.

——, and William Lewis. 2007. Multilingual structural projection across interlinear text. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, 452–459, Rochester, New York, USA.

Xue, Naiwen, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering* 11(02).207–238.

Yarowsky, David. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 189–196, Cambridge, Massachusetts, USA.

Yeh, Alexander. 2000. More accurate tests for the statistical significance of result differences. In *Proceedings of the 16th International conference on Computational Linguistics (COLING 2000)*, 947–953, Saarbrücken, Germany.

Ytrestøl, Gisle, Dan Flickinger, and Stephan Oepen. 2009. Extracting and annotating Wikipedia sub-domains – towards a new eScience community resource. In *Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theories*, 185–197, Groeningen, The Netherlands.

Zhang, Yi, and Valia Kordoni. 2010. Discriminant ranking for efficient treebanking. In *Proceedings of the 23rd International Conference on Computational Linguistics – Poster Session (COLING 2010)*, 1453–1461, Beijing, China.

——, Stephan Oepen, and John Carroll. 2007. Efficiency in unification-based n-best parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT 2007)*, 48–59, Prague, Czech Republic.

Zhou, Deyu, and Yulan He. 2008. Extracting interactions between proteins from the literature. *Journal of Biomedical Informatics* 41(2).393 – 407.

Zhou, GuoDong, and Jian Su. 2004. Exploring deep knowledge resources in biomedical name recognition. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP) 2004* , 99–102, Geneva, Switzerland.

Zhu, Muhua, Jingbo Zhu, and Tong Xiao. 2010. Heterogeneous parsing via collaborative decoding. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, 1344–1352, Beijing, China.

Zhu, Xiaojin, and Andrew Goldberg. 2009. *Introduction to Semi-Supervised Learning*. San Rafael, California, USA: Morgan & Claypool.

Appendix A

Annotation guidelines for biomedical data

The following guidelines were used by the treebankers for creating the ERGGTB corpus described in Section 5.3.1.

- In NPs with both prenominal adj and postnominal PP, the grammar already arbitrarily forces you to attach the PP to the N before attaching the Adj: [Adj [N PP]]. Since both modifiers are intersective, the semantics always comes out the same.
- In some instances it seems preferable to keep an analysis which is only wrong in some relatively trivial respect, especially since it may be difficult for a given annotator to know whether the best available analysis is fully correct or not. So accept as a good a tree which has (or might have) a minor flaw, and mark that item with a lower confidence rating.
- Appositions: use apposition only when both parts are full NPs.
- Parentheticals: apposition or not – consistently treat as parenthetical, not apposition, and attach as low as is plausible, except when the two NPs are not proper names.
- Noun vs. adj for prenominal modifier – prefer noun unless clearly adjective
- Complex proper names – use NP-HDN_NME-CPD_C rule, not NP-HDN_TTL-CPD_C
- Proper name unary rule HDN_BNP-PN_C – apply to largest span available
- Measure phrases – take measure phrase whenever offered, unless it is clearly wrong.

- Noun compounds and prenominal adjectives – follow GENIA bracketing where available; otherwise use best judgment, but where unclear, use left bracketing for nouns but attach adjectives as high as possible.
- PP attachment: defer to GENIA in cases where the ambiguity is domain-specific and cannot be determined on linguistic grounds, unless you are really sure it is wrong (but leaving GENIA aside, in general prefer high attachment for modifiers, both in NPs and in VPs or Ss).
- Multi-token GENIA expressions – use only when no good decomposed analysis exists.
- Coordination of NP vs. N-bar – prefer NP unless clearly wrong.
- Coordination bracketing – when in doubt, follow GENIA bracketing.
- Subordinate clause attachment – attach as high as possible. For attachment of scopal clausal modifiers: finite subordinate clauses attach to S if possible, while non-finite ones attach at VP.
- Slashes: treat as coordinator or preposition based on intended meaning.
- Digits: avoid ONE_DIGIT_HOUR unless the number really denotes clocktime.
- Conjunction vs. preposition for *except* – choose preposition unless clear.
- Dashes as suffixes: prefer suffix analysis, so avoid e.g. *c-minus*.
- Passive verb vs. adjective: choose verb unless clearly wrong (eg subject to degree modification).
- Choice of which *by*: avoid BY_MEANS_GER, instead choosing simple by_p (passive marker)
- Choice of using *un-* prefix rule V_V-UN_DLR: use unless obviously wrong. Prefer prefix rule instead of opaque entry if the meaning is compositional, where *un-* prefix is for reversative sense, and *re-* is for repetition sense.
- For multi-token lexemes in lexicon, prefer unless clearly wrong.
- S-initial scopal vs intersection adverbials - prefer scopal.
- for *e.g.*: prefer preposition E_G_P1
- For post-auxiliary adverbs: if not comma-marked, attach as left modifiers of following VP, but if comma-marked, attach to preceding auxiliary with VMOD rule

- For Aux-Adv-V: attach Adv as left modif of V unless Adv is *not*
- Avoid use of lex entry AND_THUS_1
- Hyphenated adjective prefix *anti-*: avoid W_HYPHEN_PLR rule
- For numbered items: use NP-CL_NUMITEM_C rule
- Add convention for commas: reject W_COMMA-NF_PLR rule when given choice. These nonformal commas appear in places where strictly speaking (on the Oxford view of punctuation) they should not. If the grammar can produce an analysis using the more prescriptive comma, we want it.
- For noun + PP-of: prefer HD-CMP analysis, not possessive *of*
- For *-PR* variants of rules like HDN-NP_APP(-PR)_C: reject -PR. The *-PR* variant in each case is intended for modifiers which are marked on both sides by a (paired) comma, while the non-paired alternative won't have matching commas (or possibly a period in place of the second comma if it's at the end of a sentence). This pairing gets confused at present in the grammar when there is some other punctuation mark at the boundary of the modifier (like a parenthesis or a double quote). So a rule of thumb is to reject the *-PR* alternative - if the grammar presents both, it probably got fooled into wrongly proposing the paired analysis
- For degree adverbs vs. pre-head modifiers: reject degree adv. Take AJ-HD whenever there's a choice. This choice is an unwelcome holdover from an earlier era where I treated most of these degree modifiers as specifiers (hence SP-HD) which were selected for by the head, but I became convinced that it's a much more productive phenomenon better treated as simple modification.
- For idiomatic lex entries: reject *_I entries
- Measure abbreviations like *mg* should be plural unless the unit is *1* or a fraction.
- Prefer left-bracketed noun compounds, but attach adjectives high, e.g. (outer brackets omitted for clarity)
  - [[N1 N2] N3] N4
  - A1 [[[N2 N3] N4] N5]
  - [[N1 N2] N3] [A4 N5]
  - A1 [[N2 N3] [A4 [[N5 N6] N7]]

Appendix B

Details of Blazing Strategies

The **MapPOS** blazing strategy from Chapter 5 used a mapping based on node labels, which we reproduce in full here. Additionally, we give details of the internal transformations applied to the node in the **RaisePremods** strategy.

GTB POSs	ERG Pref.
NN, NNS, NNP, NNPS, EX	n
VB, VBZ, VBP, VBN, VBD, VBG, MD	v
JJ, JJR, JJS	aj
RB, RBR, RBS	av
IN, RP	p

Table B.1: The correspondences between the GTB parts-of-speech and the prefix on the ERG lexical type (e.g. V_NP_LE, the type for a transitive verb, has prefix 'v') used in the **MapPOS** blazing strategy discussed in Section 5.2.2. The mapping was constructed by a manual inspection of a confusion matrix created by comparing the automatic output of a POS-tagger over sentences from WeScience with the gold-standard ERG lexical types from the corresponding manually-annotated WeScience trees.

```
foreach POS_SET in [['DT'], ['JJ', 'JJR', 'JJS', 'ADJP']]:
  foreach CONSTIT in GTB_TREE:
    if (label of CONSTIT is 'NP')
        and (CONSTIT has 2 or more subconstituents)
        and (none of ['LRB', 'RRB', 'COMMA', 'CC'] are POSs used within CONSTIT):
      let FIRST_SUBCONST := (first subconstituent of CONSTIT)
      if (label of FIRST_SUBCONST is 'NP')
          and (FIRST_SUBCONST has more than one child):
        let LEFTMOST := (first child of FIRST_SUBCONST)
        if (label of LEFTMOST is in POS_SET):
          let NEW_RIGHT := (new constituent labelled 'NP')
          let NEW_RIGHT_CHILDREN := (
              (2nd and subsequent children of FIRST_SUBCONST)
              appended to
              (2nd and subsequent children of CONSTIT)
          )
          set children of NEW_RIGHT to NEW_RIGHT_CHILDREN
          set children of CONSTIT to [LEFTMOST, NEW_RIGHT]
```

Figure B.1: Details of algorithm for **RaisePremods** (described in Section 5.2.2), after binarisation has already taken place. 'child' refers to any child of a node, while 'subconstituent' refers only to those children which are inner nodes (i.e. are not POS label). The algorithm applies first to determiners and then to adjectives as target POS sets; in each case it examines each NP node **CONSTIT** with an NP child, and if the label of the leftmost grandchild **LEFTMOST** is one of the target POSs, the node is raised to become a child of **CONSTIT**, with a new NP node **NEW_RIGHT** created as its sibling, acting as parent to the other descendants of **CONSTIT**